HÖHERE TECHNISCHE BUNDESLEHRANSTALT

Klagenfurt, Mössingerstrasse

Abteilung ELEKTRONIK

Ausbildungsschwerpunkt

Technische Informatik

Unterrichtsprojekt

WAPSCAS W@P-unterstütztes Gastronomie Abrechnungssystem

Wireless application protocol supported catering accounting system

von — Antonitsch Raimund, Hafner Dietmar

Projektbetreuer — Günter Hölzl, Johann Voglgruber

Klagenfurt, am 09.05. 2001

Inhaltsverzeichnis

1 Kurzbeschreibung des Projektes ... 3
1.1 Englische Beschreibung ... 3
1.2 Erreichte Spezifikationen ... 3
1.3 Abweichung vom Pflichtenheft ... 3
2 Pflichtenheft ... 4
2.1 Aufgabenstellung ... 4
2.2 Systemeinsatz und Systemumgebung ... 4
2.3 Funktionale Anforderungen ... 5
2.3.1 Systemstart ... 5
2.3.2 Systembetrieb ... 6
2.4 Nicht funktionale Anforderungen ... 6
2.5 Benutzerschnittstelle ... 6
2.5.1 WAP-Handy ... 6
2.5.2 Webinterface ... 6
2.5.3 Flussdiagramm für Funktionsabläufe ... 7
2.6 Dokumentation und Präsentation ... 7
2.7 Glossar und Index ... 8
3 Projektdokumentation ... 8
3.1 Marktstudie ... 8
3.2 Entwicklungstools ... 10
3.3 Lösungswege: ... 13
3.4 Projektplan / Zeitplan ... 13
3.5 Projekttagebuch ... 15
4 Systemdokumentation ... 16
4.1 Grobstruktur ... 16
4.1.1 Funktionsablauf ... 16
4.1.1.1 Dateneingabe ... 16
4.1.1.2 Datenmanipulation ... 17
4.1.1.3 Kellnerlogin ... 18
4.1.1.4 Bestellen ... 19
4.1.1.5 Abrechnen ... 19
4.1.1.6 Auftragsausgabe und Bestätigung ... 20
4.2 Feinentwurf ... 20
4.2.1 Hauptklasse: Wapscas.java ... 20
4.2.2 Servlets ... 25
4.2.3 WML – Programmierung ... 26
4.3 Dateiformate ... 27
4.4 Testdokumentation ... 27
4.5 Wartung ... 28
5 Benutzerdokumentation ... 28
5.1 Installation und Konfiguration (für Win32 Systeme) ... 28
5.2 Bedienungsanleitung für das Webinterface ... 30
5.3 Bedienungsanleitung für das WAP Interface ... 34
5.4 Mögliche Fehlermeldungen (Web- und WAP Interface) ... 37
5.4.1 Webinterface ... 37
5.4.2 WAP Interface ... 39
6 Diskussion ... 40
6.1 Probleme bei der Projektentwicklung ... 40
6.2 Erweiterungsmöglichkeiten ... 40
6.3 Vorschläge für weitere Entwicklung ... 41
7 Literaturverzeichnis ... 42
Anhang A
Anhang B

1 Kurzbeschreibung des Projektes

WAPSCAS bedeutet "Wireless Application Protocol Supported Catering Accounting System", was soviel heißt wie, W@P-unterstütztes Gastronomie Abrechnungssystem.

WAPSCAS ist ein System, dass den Bestell- und Abrechnungsvorgang in einem Gastronomiebetrieb beschleunigen soll: Jeder Kellner kann mit Hilfe eines WAP-fähiges Handys die Bestellung direkt am Tisch aufnehmen und versenden. Kurz darauf erscheint diese am Bildschirm des Küchen-PCs (per Internet werden die Daten übermittelt) und der Koch bzw. Barkeeper kann gleich mit dem Auftrag beginnen.

1.1 Englische Beschreibung

Wireless Application Protocol Supported Catering Accounting System is the name of our project. Its goal is to accelerate the working processes in a gastronomic company.

The waiter uses a WAP cellular to order food directly located at the table. After this the order is going to be displayed in the Webinterface (e.g.: PC in the kitchen). The cook or the barkeeper could start with his/her work.

1.2 Erreichte Spezifikationen

In erster Linie standen unsere eigens geschaffenen Spezifikationen im Vordergrund. Mit Stolz auf unser Projekt kann verkündet werden, dass nahezu alles unsere Spezifikationen erfüllt wurden.

Nun wird Bezug auf das Pflichtenheft (siehe Punkt2) genommen.

Die Aufgabenstellung wurde von uns weitgehend erfüllt, da es zu einer Erleichterung im Arbeitsablauf eines Gastronomiebetriebes kommt. Wir haben Gastronomiebetriebe „unter die Lupe“ genommen und können feststellen, dass der Einsatz von WAPSCAS wirklich eine Arbeitserleichterung darstellt. Auch Betriebe mit anderen Bestell- und Abrechnungssystemen (siehe Punkt3.1) sind nicht sehr viel besser als unser System.

In Sachen „Funktionale Anforderungen“ kann auf den Systemstart eingegangen werden. Der Server an sich, ist soweit automatisiert, dass er nur noch hochgefahren werden muss. Auch die Dateneingaben wurden mit Logins realisiert.

Zum Systembetrieb bleibt zu sagen, dass die Aufnahme der Bestellung, die Erfassung der Daten, die Bestätigung der Daten und die Abrechnung, genau wie im Pflichtenheft beschrieben, in die Tat umgesetzt wurden.

Auf einen weiterer Punkt, die „Einfachheit des Systems“, wurde ebenfalls Wert gelegt. Da mit einen Webbrowser nahezu jeder heutzutage umgehen kann, ist das Benutzen eines WAP Handys eine Angelegenheit. Die Bedienung am WAP Handy kann nicht verallgemeinert werden, da sie gerätespezifisch ist, doch im Großen und Ganzen auch keine Umstände bereitet.

1.3 Abweichung vom Pflichtenheft

Wie gesagt wurde nahezu alles erreicht, was wir festgelegt haben. Rein der Spezifikation bei Systemeinsatz und Umgebung legte man sich verfrüht auf etwas fest, worauf dann später noch eingegangen wird.

Die zu einsetzende Browsersoftware darf in unseren Fall nicht Netscape sein. Eine Gestaltung für diesen Browsertyp wurde nicht mehr eingebunden. Das Webinterface ist somit für den Microsoft Internet Explorer konzipiert.

Das Serverprogramm, welches unsere Daten für die Datenbank verwaltet, wurde wie bereits erwähnt zu früh festgelegt. Hierbei kam es zu einer großen Wende. Es wird angeführt, das die

Datenverwaltungsprozesse mit einen CGI-Script, welches in PERL programmiert wird, zu realisieren sind. Die Datenverwaltung wurde dann auf JDBC mit Java Servlets umgestellt.

Die erwähnte Monats- und Tagesabrechnung wurde ein „Opfer der Zeit“. Die Einbindung und Implementierung stellte eigentlich kein Problem dar und kann als eine Erweiterungsmöglichkeit (Punkt6.2) gesehen werden.

Das Flussdiagramm der Funktionsabläufe wurde auch nicht ganz so umgesetzt wie es angeführt ist. Nach einer Bestellung vom Handy aus, muss man sich nämlich nicht erneut einloggen. Die Begründung liegt darin, da es nicht für sinnvoll erklärt wurde, sich ein 2. Mal einzuloggen, da dadurch einige Zeit verloren geht. (Login ist mit Verbindungsaufbau verbunden).

2 Pflichtenheft

2.1 Aufgabenstellung

- *Softwareprojekt mit WAP Technologie*
- *Einsatzbereich: Gastronomie*

Grundsätzlich befasst sich das Projekt mit der WAP Technologie. Bei unserem Projekt liegt das Einsatzgebiet im Bereich der Gastronomie.

Ein Kellner eines Gastronomiebetriebes soll seine Bestellungen und Abrechnungen per WAP-Handy aus übermitteln können. Am lokalen Standort (Küche mit PC, der online ist) werden die Daten des Kellners aufgenommen und verwaltet. Dadurch sollten sich die Wartezeiten für die Gäste verkleinern.

2.2 Systemeinsatz und Systemumgebung

Folgende Dateneingabe vom Kunden ist erforderlich, damit unser System überhaupt eingesetzt werden kann.

1. Speisen und Getränke
2. Preise
3. Tischanzahl

Hardware:

WAP Handy:
Im allgemeinen sollte jedes WAP Handy verwendet werden können, sofern es über einen WML-Browser ab der Version 1.1 verfügt. Hilfreich wäre jedoch ein großes Display, um eine bessere Übersicht zu gewährleisten.

Browser PC:
Der eingesetze PC muss über keine besondere Hardware verfügen, lediglich eine Online-Verbindung muss vorhanden sein. Es kann auch ein Notebook verwendet werden.

Server PC:
Dieser PC muss ebenfalls über eine Online Verbindung und ausserdem über ausreichend viel Speicherplatz verfügen. Von Vorteil wäre es, wenn die Online Verbindung als Standleitung realisiert wäre.

Software:

WML Browser am Handy:
Der Browser, welcher am WAP Handy läuft, muss lediglich die Version 1.1 aufweisen. Eine weitere Konfiguration des WAP Handys muss nicht erfolgen. (nur mit Standardportal des GSM-Netzproviders).

Browsersoftware am PC:
Eingesetzt können ab Internet Explorer 4.0 und Netscape Communicator 4.5 werden. Es ist nur wichtig, dass die Java Virtual Machine bzw. Java-Plugin installiert ist.

Serverprogramme:
Zu verwenden sind APACHE Webserver, der die Verbindung zum WAP Handy herstellt, dann ein PERL Interpreter für die CGI-Skript Bearbeitung sowie eine Datenbank, wo sich die eigentlichen Daten befinden.

2.3 Funktionale Anforderungen

2.3.1 Systemstart

1. Starten des Datenbankservers
Der Server-PC muss lediglich eingeschaltet werden und startet dann möglichst automatisch die Serverdienste.

2. Dateneingabe vom Kunden
Bevor das System gestartet werden kann, müssen vom Kunden die oben beschriebenen Daten eingegeben werden. Dies sollte über das Webinterface erfolgen und muss nicht jedes Mal eingegeben werden, da sich die Speisekarte nicht ständig ändert.

3. Einwahl mit dem WAP-Handy
Nachdem das WAP-Handy entsprechend konfiguriert wurde, kann man sich ins WAP-Portal einwählen. Um sich in unser System einloggen zu können, muss man anschließend Benutzername und Kennwort eingeben.

2.3.2 Systembetrieb

1. Aufnahme der Bestellung
Um mit der Bestellung zu beginnen, muss zunächst die Tischnummer am Handy eingeben werden. Eine hierarchische Menüführung am Handy ermöglicht danach eine unkomplizierte Bestellung.

2. Erfassung der Daten
Die vom Handy aus bestellten Getränke und Speisen werden am PC, welcher sich in der Küche befindet, im Webinterface angezeigt.

3. Bestätigung der Daten
Um dem Küchenpersonal eine Möglichkeit zu geben, seine Arbeit als „abgehakt“ zu markieren, kann im Webinterface eine Bestätigung erfolgen.

4. Abrechnung
Nachdem vom Kellner die Tischnummer eingegeben wurde, erscheint am Display der Rechnungsbetrag.

Nach erfolgter Abrechung werden die Bestellungsdaten des jeweiligen Tisches gelöscht. Sämtliche Bestellungen werden zeitgleich mit ihrer Aufnahme in eine Datenbank gespeichert. Dies dient der Tages- bzw. Monatsabrechnung.

2.4 Nicht funktionale Anforderungen

- *unkomplizierte und einfache Anwendung*
- *Schnelligkeit und Betriebssicherheit*

An erster Stelle steht die unkomplizierte und einfache Bedienung am WAP Handy sowie am PC. Der Benutzer (Kellner, Koch, ...) sollte nicht zuerst ein dickes Handbuch lesen müssen, sondern nach einer kurzen Einschulung fähig sein, seine Arbeit zu verrichten. Neben Schnelligkeit und Betriebssicherheit sollte auch darauf geachtet werden, dass wegen der geringen Größe des Displays nur soviel Information darauf erscheint, wie nötig ist.

2.5 Benutzerschnittstelle

2.5.1 WAP-Handy

Am WAP-Handy erfolgt die Bestellung. Die Menüführung sollte so ausgerichtet sein, dass der Kellner die Getränke und Speisen möglichst schnell auswählen kann. Nach erfolgter Eingabe muss anschließend die Bestellung bestätigt werden.

2.5.2 Webinterface

Die aufgenommene Bestellung wird in der Küche bzw. Bar am Webinterface immer wieder (automatische Aktualisierung der Seite) angezeigt. Übersichtlichkeit und eine einfache Bedienung stehen dabei an erster Stelle.

2.5.3 Flussdiagramm für Funktionsabläufe

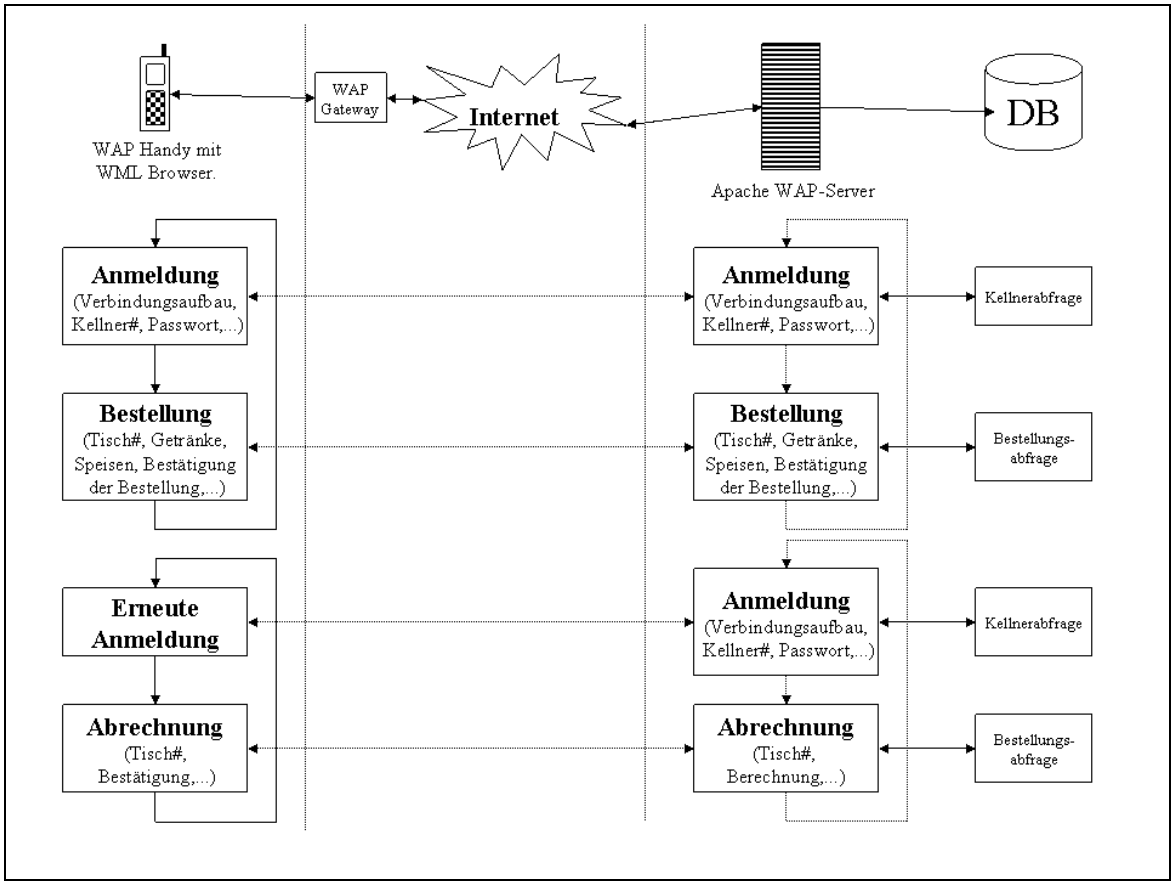

Abbildung 1

2.6 Dokumentation und Präsentation

- *Dokumentationsumfang*
- *Richtlinien zur Erstellung*
- *Benutzerhandbuch*
- *Präsentationen*

Die Dokumentation liefert grundsätzlich Information zu unserem Projekt. Sie besteht aus zwei Teilen: Allgemeiner Teil (Verwendete Programmiersprachen, eingesetzte Software und WAP Technologie) und spezifischer Teil (Entwicklung, Programmierung, ...). Die Dokumentation ist außerhalb der Unterrichtszeit zu erstellen; im Labor dürfen nur Korrekturen durchgeführt werden. Die Einhaltung der Dokumentationsrichtlinien ist zu beachten. Verwendung der Software WinWord; Erstellen des Inhaltsverzeichnisses; abschnittsweise Gliederung; Kopf- und Fußzeile.
Die Richtlinien sind in speziellen Dokumenten vorgegeben
Zeitgleich mit den 4 Milestones (Kalenderwoche 45, 4, 17 und Diplomarbeitabschluss) gibt es die entsprechenden Präsentationen, die den Projektstatus wiederspiegeln sollten. Die erste Präsentation entspricht einem Kurzreferat. Dieses sollte das Plichtenheft, Lösungsweg und Projektplanung beschreiben. Die weitere Präsentation enthält bereits die erste Version der Dokumentation. Zusätzlich ist eine Homepage zu entwickeln und zu präsentieren.

2.7 Glossar und Index

WAP	Wireless Application Protocol
Online Verbindung	Verbindung zum Internet
Browser	Programm zum Darstellen von Internetseiten
Handy	Mobiltelefon
Milestones	Meilensteine (best. Zeitpunkte)
Webinterface	Benutzerschnittstelle im Browserfenster
APACHE Webserver	Ein Webserver namens Apache
CGI	Common gateway interface
Perl	Spezielle Skript Programmiersprache
Java	Plattformunabhängige Programmiersprache

3 Projektdokumentation

3.1 Marktstudie

Hierbei wird auf die bereits bestehenden mobilen Bestellsysteme mit deren Vor- und Nachteilen in Bezug auf unser Projekt eingegangen.

Die meisten restlichen Anbieter haben sich dabei auf eine Funkverbindung spezialisiert. Gastronomiebestellsysteme dich jedoch die Vorteile und Funktionalität des Internets verwenden, sind relativ selten. In den folgenden Beispielen sind kommerzielle Gastronomiebestellsysteme angeführt.

1. **Firma Koscheck Automatisierungstechnik:** (http://home.t-online.de/home/koscheck/start.htm)

Diese Firma hat ein mobiles Abrechnungssystem entwickelt. Dazu gehören ein drahtloses Gerät und ein sog. Ferndrucker. Am Ferndrucker ist dann ein Netzwerkanschluss vorhanden, der damit mit einem PC verbunden werden kann. Weiters kann optional eine sog. SQL Box erworben werden, mit der eine Verbindung vom PC hergestellt werden kann. Sie wird am PC an der seriellen Schnittstelle angeschlossen. Es wird eine Software mitgeliefert, mit der die Abrechnungen vorgenommen werden.

Vorteile:

- Die Bestellung erfolgt mittels Kurznummern.
- Die Rechnung kann direkt ausgedruckt werden.
- Am Abrechnungsgerät ist ein Chipkartenleser integriert (Kreditkarte).

Nachteile:

- Zu hohe Anschaffungskosten (siehe Vergleich)
- Keine direkte Verwaltung per Internet oder ähnlichen Medien.
- Die Verwaltung der Daten erfolgt nicht datenbankorientiert.

2. **GastroFix** (http://www.gastrofix.de/manual/inhalt.htm)

Diese Firma hat ein ähnliches Angebot, wie zuvor beschrieben. Die Verwaltung der Daten, erfolgt in dem Fall jedoch über PCs. Dabei wird eine richtige Topology mit mehreren Servern und Arbeitsplätzen verwirklicht. Die Bestellung erfolgt per PC oder Handygerät. Die Verbindung untereinander erfolgt einerseits per Funk und andererseits per Netzwerkverbindung. Am Bestell-PC können Bondrucker angeschlossen werden.

Vorteile:

- Die Bestellung erfolgt relativ einfach.
- Bei Großbetrieben sehr sinnvolle Strukturierung
- Wartbarkeit des Systems

Nachteil:

- Wie zuvor die Anschaffungskosten.
- Ziemlich umfangreiches System, eine gewisse Einschulungszeit benötigt.
- Kein ansehnliches Interface zum Bestellen.

Kostenvergleich WAPSCAS – Koschek Automatisierungstechnik

WAPSCAS	Anschaffungskosten für: Server PC*1 Küchen PC*2 WAP Handy*3	8000+ 7000+ 1000+ **16000.-**
Koschek Automatisierungstechnik	Anschaffungskosten für: Minicash SQL Box	11200+ 20500+ **31700.-**
WAPSCAS	Betriebskosten / monatl. für: W@P Gebühr *4 Onlinekosten *5	 400+ 700+ **1100.-**
Koschek Automatisierungstechnik	Betriebskosten / monatl.	keine Angabe
Koschek Automatisierungstechnik	Wartungskosten	keine Angabe

*1... PC mittlerer Ausstattung
*2... ein etwas schwächerer bzw. älterer PC kann verwendet werden (internetfähigkeit muss nur gegeben sein)
*3... Preis entspricht bei Erstanmeldung eines GSM Anbieters (konkret auf ein Nokia Model bezogen)
*4... Tarifmodel von ONE herangezogen.
*5... Flatrateonlineanbindung von Telekabel.

3.2 Entwicklungstools

Nokia ToolKit:

Ein sehr komfortables Programm zur Erstellung von WML Seiten ist das Nokia Toolkit. Zu dieser Entwicklungsumgebung gehören ein Editor (für WML oder WML Script) sowie ein Simulator (ein Nokia Handy wird simuliert), der kompilierte WML Seiten anzeigen kann. Sehr nützlich ist die Funktion des Debuggings, da erstens alle auftreten Fehler mit entsprechender Fehlerstelle und zweitens eine Fehlerbeschreibung angezeigt werden. Der Simulator unterstützt die Anzeige von WML1.1 Seiten, Bildern (WML Bitmaps), Tabellen und WML-Script.

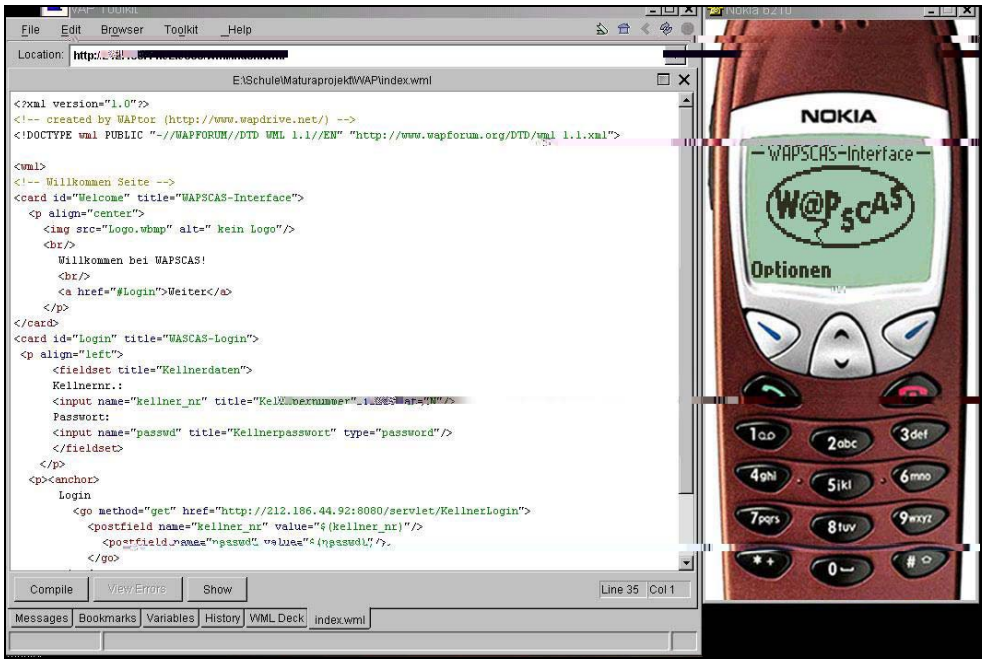

Abbildung 2

Zur Erstellung von so genannten WML Bitmaps wird ebenfalls ein eigener Editor mitgeliefert, der in das Toolkit mit integriert ist sowie Bildformate wie JPEG und GIF können in WBMP umgewandelt werden.

Das Toolkit ist in Java programmiert und verlangt somit das Java Runtime Environment (JRE) welches mit dem Programm mitgeliefert wird und automatisch installiert wird.

Weitere Funktionen:

Verlauf: Eine History der zuletzt angezeigten Seiten am Simulator wird mitgeschrieben. Sie kann manuell wieder gelöscht werden und dient dazu, eine „Zurück“ Funktion am Handy zu simulieren.

Bookmarks: Damit ist es möglich Bookmarks von URLs zu speichern. Eine Verwaltung von mehrere WAP Adressen ist somit komfortabel möglich.

WAPTor:

Dieses Programm wird beim Anbieter unseres WAP-Space angeboten und empfohlen. Es umfasst einen WML Editor und einen Simulator, der die erstellten WML Seiten anzeigen kann. Das Editor ist ähnlich einem Textverarbeitungsprogramm aufgebaut und verfügt somit über eigene Icons für

bestimmte Befehle und die Formatierung. Ein Nachteil von diesem WML Editor ist, das die WML Seiten nicht richtig simuliert werden. Man kann davon ausgehen, das eine erstellte WML Seite, welche mit dem Simulator betrachtet wird, von einer auf einen wapfähigen Handy dargestellten Seiten abweichen wird.

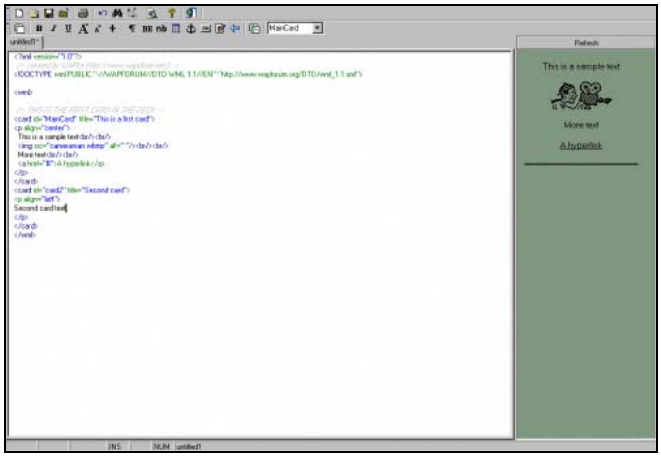

Abbildung 3

Apache WebServer (http://www.apache.org):

Diese Software ist einer der verbreitesten Webserver der heutigen Zeit. Nahezu 65% aller weltweit eingesetzten Webserver basieren auf dem Apache Webserver. Ein Webserver ist ein Programm mit dem es ermöglicht wird Internetseiten (oder sog. URLs) entweder in einen Dateinamen umzuwandeln und diese Datei per Internet an den Benutzer (der, diese Seite angefordert hat) zurückzuschicken. Weiters kann die URL auch eine Programmnamen umwandeln und z.B: das Ergebnis an den Benutzer zurückschicken.

Erwartungen an einen Webserver:

- Schnelligkeit (Verarbeitung von vielen Anfragen)
- Multitasking (Verarbeitung mehrer Anfragen zugleich)
- Fehlerbehandlung (Bestimmte Fehler anzeigen)
- Authentifizierung (Zugriffsbeschränkungen erstellen)
- Sicherheit (Gegenüber Hackerattacker)

Der Apache Server ist ein Project der ASF (Apache Software Foundation) und ist zum größten Teil plattformunabhängig. Weiters ist die Software Freeware (Open Source) und wurden von vielen Freiwilligen verstreut über die ganze Welt entwickelt. Nicht nur Entwicklng, sondern auch die Weiterentwicklung sowie Bugbehebung und Verbesserungen unterliegen dem Apache Project.

Tomcat Server (http://jakarta.apache.org/tomcat/):

Diese Software ist Teil des Jakarta Project. Der Tomcat Server beinhaltet die Implementation für Java Servlets 2.2 und Java Server Pages 1.1 . Tomcat ist für diese Technologien als Referenz anzusehen.

Adobe GoLive (http://www.adobe.com/golive):

Dieses Programm ist ein HTML Editor mit dem komfortabel professionelle Webseiten erstellt werden können. Grundsätzlich ist der Aufbau einer Internetseite auf Layout, HTML Code und Preview zurückgeführt. Im Layout Teil, können Elemente (Bilder, Tabellen, Textformatierungen) vorgenommen werden. Der HTML Code Bereich zeigt farblich hervorgehoben (siehe Abbildung) die verschiedenen HTML Befehle. Dies ist hilfreich bei der Fehlersuche. Im Preview kann die Seite dann angeschaut werden.

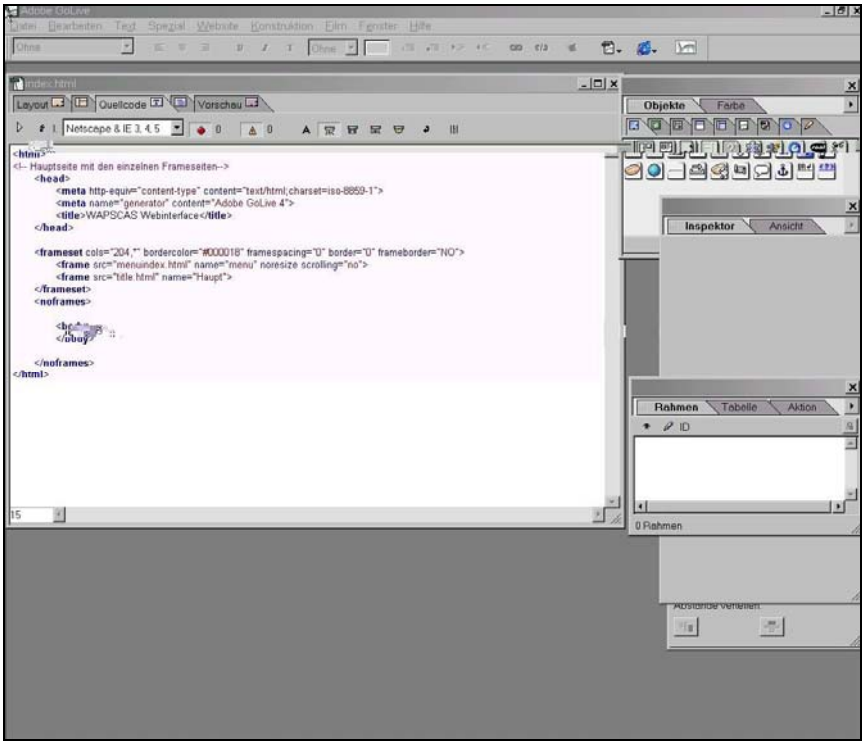

Abbildung 4

MySQL Datenbank:

MySQL ist eine echte Multi-User, Multi-Threaded SQL Datenbank und wird von allen großen Providern oder auch Suchmaschinenbetreibern eingesetzt. MySQL ist eine Client/Server Implementierung, die aus einem Server-Dämon mysqld und vielen Client Programmen, sowie Bibliotheken für PERL, PHP/3, PHP/4 sowie ASP besteht.

SQL ist eine standardisierte Datenbanksprache, die das Speichern, Updaten und den Zugriff auf Informationen erleichtert. MySQL ist äußerst schnell und flexibel genug, um sogar Bilder und Log-Dateien darin abzulegen. In der Praxis ist MySQL sehr viel schneller, als z.B. ORACLE oder INFORMIX.

Die wichtigsten Eigenschaften von MySQL sind Geschwindigkeit, Stabilität und einfache Bedienbarkeit. MySQL wurde ursprünglich entwickelt, weil auf TCX (dem Server der Entwickler) ein SQL Server benötigt wurde, der sehr große Datenbanken handeln konnte, und zwar um eine Größenordnung schneller, als die Datenbankhersteller damals liefern konnten. MySQL ist nun seit 1996 auf vielen Tausend Sites im Internet und Intranet im Einsatz und erfreut sich hier wachsender Beliebtheit.

Der Grundstock, um den MySQL herum gebaut worden ist, ist eine Liste von Routinen, die sich im täglichen Einsatz seit Jahren bewährt haben. Obwohl MySQL dauernd weiter entwickelt wird, hat es zu jedem Zeitpunkt der Entwicklung stets zuverlässig und stabil gearbeitet.

SUN Java Developer Kit:

Das Java Development Kit umfaßt die Werkzeuge, die unbedingt nötig sind, um Programme in der Sprache Java übersetzen und auszuführen: Den Bytecode - Übersetzer *javac*, den Bytecode - Interpreter *java*, den Debugger *jdb* und das Dokumentations-Werkzeug *javadoc*, sowie ein paar weitere Werkzeuge.

3.3 Lösungswege:

In der Phase der Projektplanung sowie in der Phase eine Projektspezifikation zu erstellen, wurden mehrere Lösungswege durchdacht und schlussendlich mussten wir uns auf einen Weg einigen.

Die verschiedenen Lösungswege beziehen natürlich auf unsere drei Hauptkomponenten unseres Projektes: WAP – Handy, Browser PC, Server PC.

WAP Handy

Der Einsatz eines WAP Handys wurde nicht diskutiert, da ein wapfähiges Handy sowieso eingesetzt werden muss, weil WAP unser Verbindungsmedium (Internet) ausnutzt. Die Bestellungen vorzunehmen mit Hilfe eines WAP Handys erschien uns als sehr hilfreich, da die Programmierung von WML Seiten sehr ähnlich der HTML Programmierung ist. Ein anderer Lösungsweg für die Bestellfunktion oder Bestellaufnahme würde womöglich per Funk realisiert werden.

Browser PC

Der Browser PC ist auch eine wichtige Hauptkomponente, bei der es zuerst zu einigen Unstimmigkeiten bezüglich der Programmierung gab. Im Browser wird unser Webinterface angezeigt, welches aktuelle Bestellungen anzeigt. Es wurde auch festgelegt, dass die aktuellen Bestellungen in einem Webbrowser angezeigt werden und somit HTML Programmierung von Nöten ist.

Zunächst gingen wir davon aus, die Oberfläche im Webinterface nicht auf HTML aufzubauen, sondern auf eine JAVA Oberfläche. Es wurden auch Versuche damit gemacht, doch wir entschieden uns, zu HTML zurückzukehren, da eine JAVA Oberfläche nur einen erhöhten Ressourcenbedarf erzielen würde.

Da nun HTML Programmierung eingesetzt wurde, war man in Sachen Layout, Eingabeabfragen und weiteren Dingen um einiges flexibler.

Server PC

Lösungswege, die den Server betrafen gab es anfangs einige, weil es darum ging, die Datenverwaltung besonders effizient zu gestalten. Nun war am Anfang des Thema CGI –Script mit PERL relevant (siehe Punkt1.2). Davon kamen wir jedoch wieder ab, da erstens das zweite Softwareprojektteam JDBC einsetzt und Java Servlet besser dazu passen.

3.4 Projektplan / Zeitplan

Der von uns geschaffene Zeitplan, kann auch als Projektplan angesehen werden. Es gab viele Änderungen während des Projektes. Diese wurden dann in den Zeitplan übernommen.

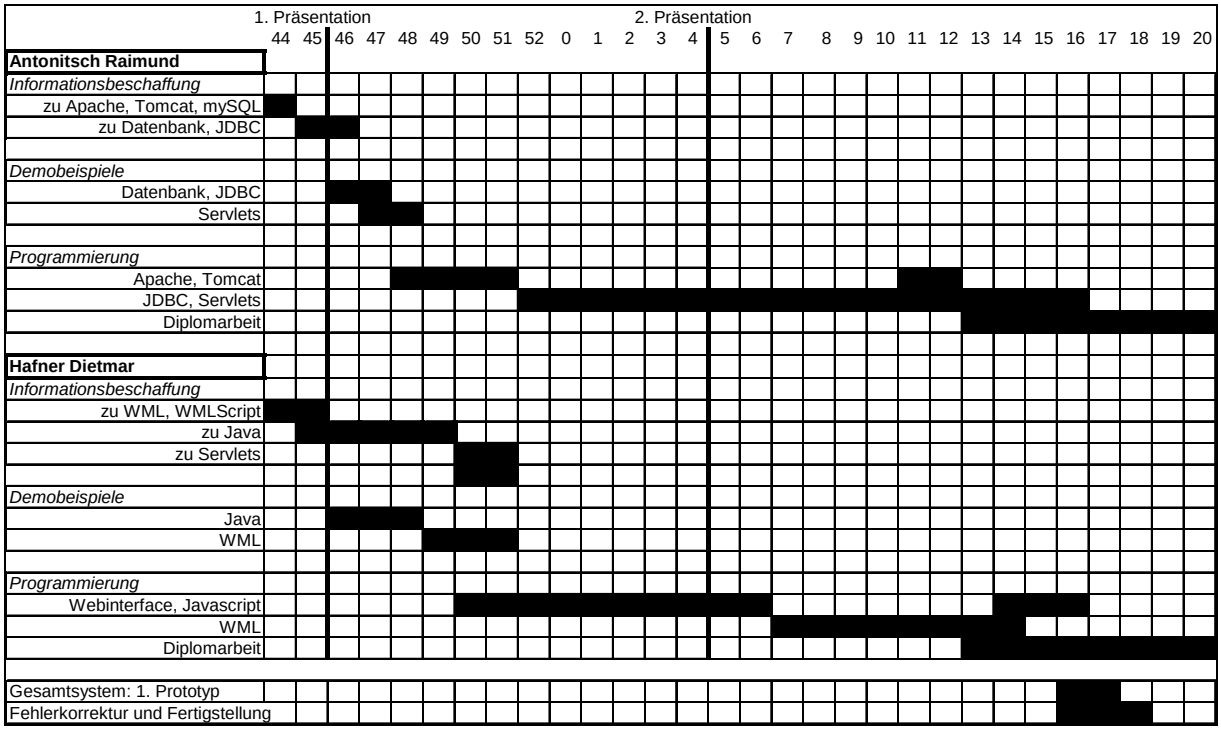

Abbildung 5

Kritische Teile des Zeitplan waren in die ersten Wochen der Projektarbeit. Ständig wurde etwas verändert.

3.5 Projekttagebuch

Woche	Antonitsch	Hafner
43	Arbeitseinteilung	Arbeitseinteilung
44	LINUX Installation & Konfiguration	WML Editor auswählen, sowie Einarbeiten mit Visual Age
45	MySQL Installation & Konfiguration	Visual Age
46	Datenbank erstellen	Visual Age
47	JDBC-Programmierung (erste Versuche)	Visual Age
48	JDBC-Programmierung	WML WAPTor Programmierung
49	Tomcat Installation&Konfiguration	Java Servlets Grundlagen, WAP Seiten Gestaltung
50	Kellner eintragen	Webinterface Prototyp, Umstellung: weg von Visual Age
51	Kellner ausgeben	Webinterface
52	Kellner löschen	Verbesserung des Webinterface
1	Speisen eintragen	Verbesserung des Webinterface, Datenbankankopplung
2	Speisen ausgeben, Speisen löschen&updaten	Verbesserung des Webinterface, Fehlerabfragen
3	Speisen ausgeben, Speisen löschen&updaten	Verbesserung des Webinterface
4	Aufträge ausgeben	Verbesserung des Webinterface
5	Auftragsverwaltung	Verbesserung des Webinterface
6	Aufträge bestätigen	Verbesserung des Webinterface
7	Datenbankänderung (Primärschlüsseländerung)	WML Seitenprogrammierung
8	WAP-Servlets	WAP Versuche
9	WAP-Servlets	Apache Server für WAP einrichten
10	Abrechnung	Webinterfaceanpassung
11	Abrechnung	WML Seiten nun doch großteils dynamisch erstellen lassen
12	Unbezahlte Aufträge	Serverarbeiten zu Hause
13	Abrechnung (Korrektur)	Serverarbeiten zu Hause
14	Abrechnung (Korrektur)	Neues und jetziges Webinterface
15	Kellnerlogin	Verbesserung sowie Feinheiten optimieren, Auftragsreset
16	Auftragreset, Schönheitskur	Fertigstellung des endgültigen Webinterfaces
17	Test des Gesamtsystems	WAP Versuche + Zusammenspiel mit Webinterface testen
18	Fehlerkorrektur	WAP Versuche + Zusammenspiel mit Webinterface testen
19	Diplomarbeitanfänge	Diplomarbeit - benötigte SW feststellen
20	Diplomarbeit	Diplomarbeitsanfänge

4 Systemdokumentation

4.1 Grobstruktur

4.1.1 Funktionsablauf

Im Allgemeinen besteht WAPSCAS aus 6 Funktionen:

- Dateneingabe
- Datenmanipulation
- Kellnerlogin
- Bestellung
- Abrechnung
- Auftragsausgabe

Dieser Funktionsablauf soll zeigen, wie WAPSCAS diese 6 Funktionen abarbeitet.

4.1.1.1 Dateneingabe

Bevor WAPSCAS betriebsbereit ist, ist es notwendig, die MySQL-Datenbank mit Daten zu fühlen. Zu diesen Daten zählen zunächst die Kellnerdaten (Vorname, Nachnahme, Passwort) und die Speisekarte (Name, Art, Preis). Die Eingabe der Daten erfolgt ausschließlich im Webinterface.

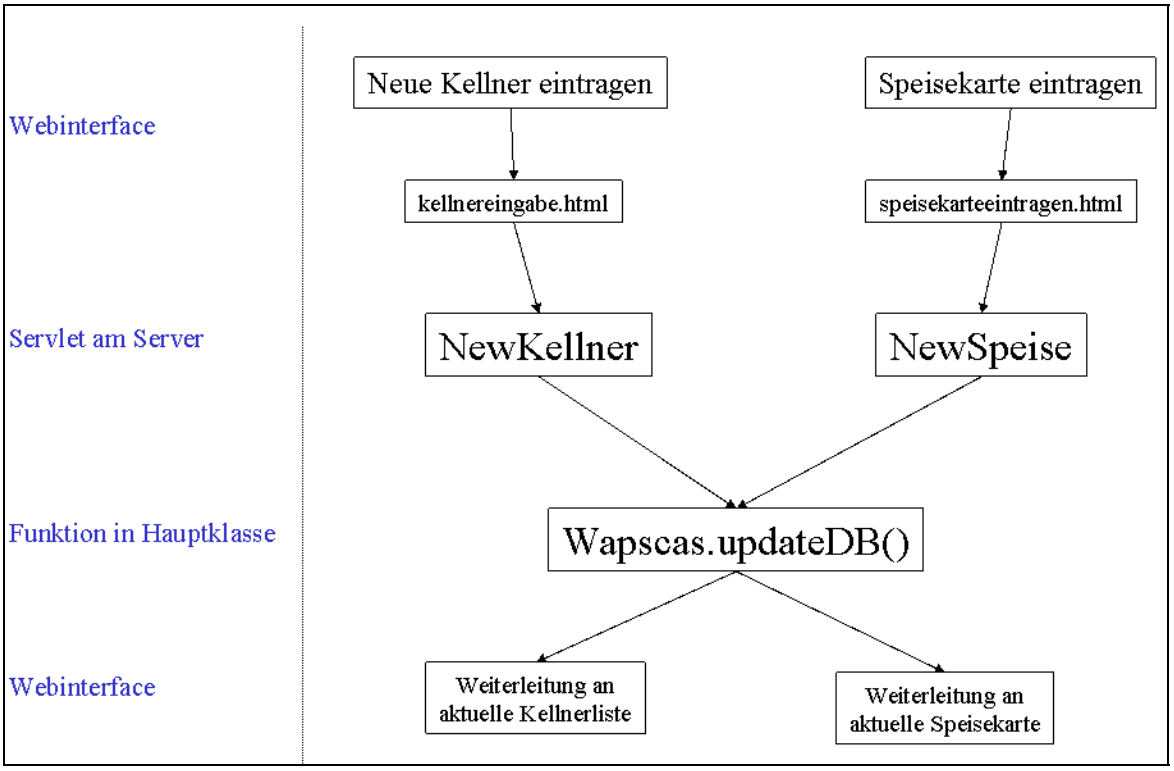

Abbildung 6

Eingabe der Kellnerdaten

Über den Menüpunkt „Neue Kellner eintragen“ gelangt man zum Eingabeformular (kellnereingabe.html). Nachdem die Daten eingetragen wurden, startet der Benutzer mit dem „Eintragen“ – Button das NewKellner – Servlet (NewKellner.java). Dieses Servlet wiederum startet

die Funktion updateDB in der Wapscas – Klasse (Wapscas.updateDB), welche die zuvor eingetragenen Kellnerdaten in die Datenbank schreibt. Anschließend erscheint am Webinterface die aktuelle Kellnerliste.

Eingabe der Speisekarte

Die Eingabe der Speisekarte verläuft ziemlich ähnlich wie die Kellnereingabe. Über den Menüpunkt „Speisekarte eintragen“ gelangt man zum Eingabeformular (speisekarteeintragen.html). Die Daten werden eingegeben, das Servlet NewSpeise (NewSpeise.java) gestartet und anschließend wird wieder zur aktuellen Speisekarte weitergeleitet.

4.1.1.2 Datenmanipulation

Natürlich ist es auch möglich Daten, die bereits in der Datenbank stehen, zu löschen oder zu ändern.

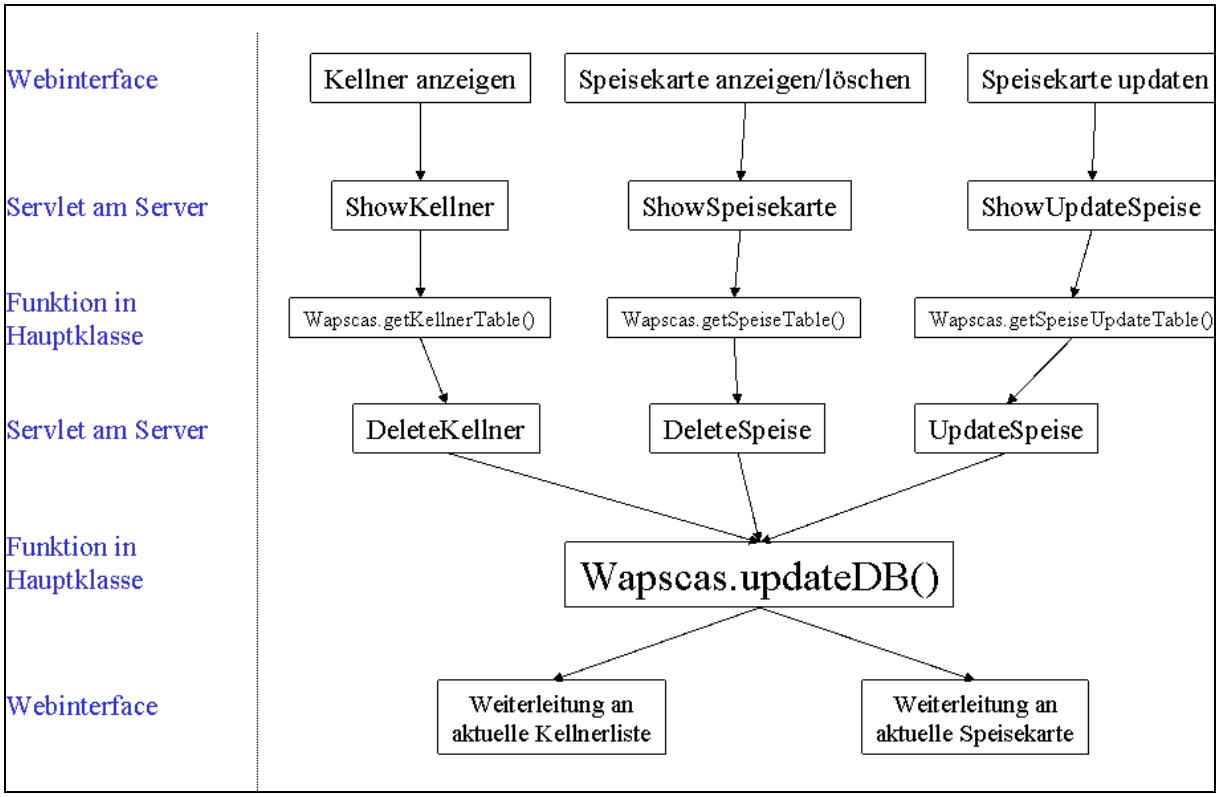

Abbildung 7

Kellner entfernen

Mit einem Klick auf den Menüpunkt „Kellner anzeigen“ wird das ShowKellner – Servlet gestartet. Dieses Servlet startet die Funktion getKellnerTable in der Hauptklasse (Wapscas.getKellnerTable). Anschließend erscheint im Webinterface die aktuelle Kellnerliste. Jeder Kellnereintrag lässt sich über ein Radiobutton auswählen. Nachdem ein Kellner ausgewählt wurde, kann der Benutzer mit dem

„Löschen“ – Button das DeleteKellner – Servlet starten. Dieses wiederum startet die Funktion updateDB in der Hauptklasse, welche den Kellner aus der Datenbank entfernt.

Speise/Getränk entfernen

Unter dem Menüpunkt „Speisekarte anzeigen“ wird das ShowSpeisekarte – Servlet gestartet. Dieses Servlet startet die Funktion getSpeiseTable. Anschließend erscheint die aktuelle Speisekarte im Webinterface. Mit den Radiobuttons kann man die zu löschende Speise auswählen und mit dem Löschen – Button aus der Datenbank entfernen.

Speisekarte updaten

Will der Benutzer nun den Preis einer Speise bzw. Getränkes ändern, kann er dies unter dem Menüpunkt „Speisekarte updaten“ tun. Das ShowUpdateSpeise – Servlet liefert dem Webinterface, mit Hilfe der Hauptklassenfunktion getSpeiseUpdateTable, die aktuelle Speisekarte zurück. Radiobuttons dienen zur Auswahl der Speise und ein Textfeld zur Eingabe des neuen Preises. Mit dem Updaten – Button wird das UpdateSpeise – Servlet gestartet. Dieses führt ein Datenbankupdate (Wapscas.updateDB) aus und zeigt anschließend die geänderte Speisekarte an.

4.1.1.3 Kellnerlogin

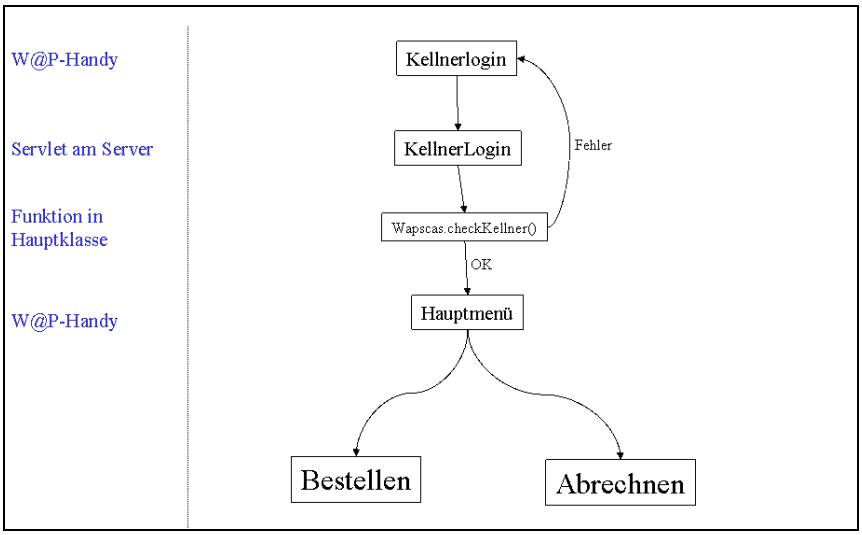

Abbildung 8

Auf der WAP – Startseite gelangt man zuallererst zum Kellnerlogin. Hier kann der Kellner seine Nummer und sein Passwort eingeben und das Servlet KellnerLogin starten. Dieses Servlet übergibt die Kellnerdaten an die Hauptklassenfunktion checkKellner und leitet, je nachdem, ob richtig oder falsch eingeloggt wurde, zum Wapscas-Hauptmenü oder erneut zum Kellnerlogin weiter.

4.1.1.4 Bestellen

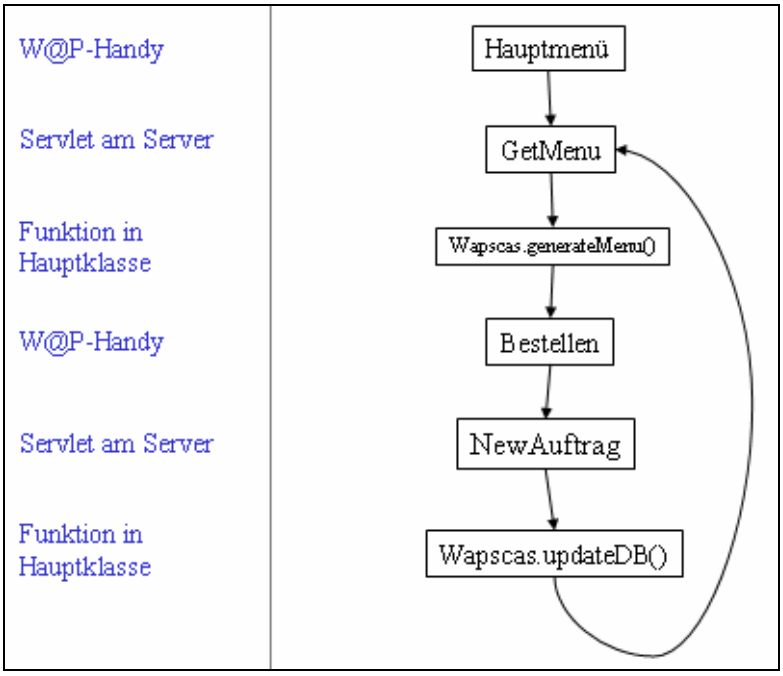

Abbildung 9

Vom Hauptmenü aus wird über den Menüpunkt „Bestellen“ das GetMenu – Servlet gestartet. Die Funktion generateMenu erstellt mit Hilfe der Datenbank ein Bestellmenü. Nachdem die Bestellung eingegeben und abgeschickt wurde, startet das NewAuftrag – Servlet die Funktion updateDB und somit steht der aufgenommene Auftrag in der Datenbank. Anschließend gelangt man wieder zum Bestellmenü.

4.1.1.5 Abrechnen

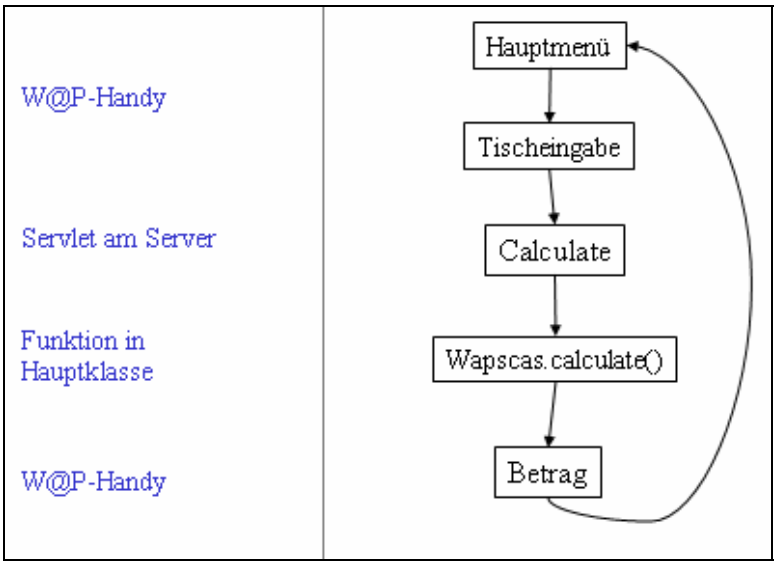

Abbildung 10

Der Menüpunkt „Abrechnen“ im Hauptmenü dient, wie der Name schon sagt, zur Abrechnung. Hier kann der Kellner die Tischnummer eingeben. Das Calculate – Servlet startet die Hauptklassenfunktion calculate. Diese Funktion rechnet alle unbezahlten Aufträge der eingegebenen Tischnummer

zusammen. Dieser Betrag wird dann am Handydisplay angezeigt. Danach gelangt man wieder zum Hauptmenü.

4.1.1.6 Auftragsausgabe und Bestätigung

Über den Menüpunkt „Aufträge anzeigen“ kann sich der Koch die aktuellen Aufträge anzeigen lassen. Das ShowAuftraege – Servlet startet die Funktion getAuftragTable. Die Aufträge werden in Abständen von 10 Sekunden aktualisiert. Will der Koch einen Auftrag als fertig markieren, muss er nur den Radiobutton der Speise markieren und anschließend auf „Fertig“ klicken. Das AuftragFinished – Servlet führt mit der Funktion updateDB ein Datenbankupdate aus und zeigt wieder die aktuellen Aufträge an.

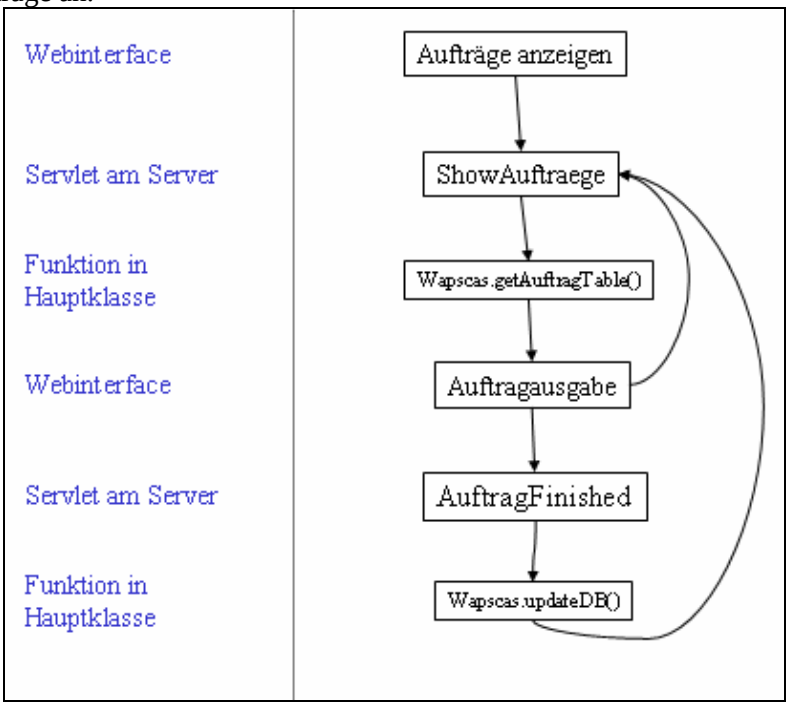

Abbildung 11

4.2 Feinentwurf

Wie die oben erwähnten Funktionen nun im detail funktionieren wird nun näher erläutert. Anfangen möchten wir mit der Hauptklasse:

4.2.1 Hauptklasse: Wapscas.java

Diese Klasse enthält Methoden, die hauptsächlich von den Servlets gebraucht werden. Die Methoden sind alle static, da es nicht notwendig ist eine Instanz von Wapscas zu erzeugen. Exceptions werden alle "gethrowed", damit die Servlets sie ggf. ausgeben können.

```
public class Wapscas
{
	// statische Variablen, die immer gebraucht werden
	static String driver = "org.gjt.mm.mysql.Driver"; //MySQL-JDBC-Treiber
	static String url = "jdbc:mysql://127.0.0.1/WAPSCAS"; //MySQL-DB-Url
	static String user = "root"; //MySQL-Username
	static String passwd = ""; //MySQL-Passwort
	static String server = "http://212.186.92.44:"; //Server-IP
	static String port1 = "9080"; //Webserver-Port
	static String port2 = "8080"; //Tomcat-Port
```

Methode: getResultSet

Die Methode getResultSet bekommt eine SQL-Query, stellt eine Verbindung zu der MySQL-Datenbank her, führt die Query aus und liefert das ResultSet zurück. Sie wird immer dann gebraucht, wenn Servlets oder andere Funktionen (calculate), Daten aus der Datenbank lesen wollen.

```
public static ResultSet getResultSet(String query)
        throws SQLException, ClassNotFoundException
{
        Class.forName(driver);
        Connection con = DriverManager.getConnection(url,user,passwd);
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        con.close(); // Schließen der Connection
        stmt.close(); // Schließen des Statements
        return(rs); // Rückgabe des ResultSets
}
```

Methode: headWithTitle

headWithTitle ist eine Hilfsmethode für Servlets. Sie liefert den Anfang einer HTML-Datei zurück: DOCTYPE, Kopf, Titel und den Stylesheet-Link. Wenn HTML-Code in den Servlets erzeugt wird, ist sie die erste Funktion, die Code dafür liefert.

```
public static String headWithTitle(String title)
{
        String DOCTYPE = "<!DOCTYPE html PUBLIC \"-//W3C//Dtd html 4.0 " +
                                "Transitional//EN\">\n";
        return(DOCTYPE +
                "<html>\n" +
                "<head>\n" +
                "<link rel=\"stylesheet\" type=\"text/css\"" +
                "href=\"" + server + port1 + "/webinterface/css/format.css\">\n" +
                "<title>" + title + "</title></head>\n" +
                "<body text=\"yellow\">\n");
}
```

Methode: WMLHead

WMLHead liefert den WML-Kopf zurück. Sie wird bei jedem WML-Code benötigt.

```
public static String WMLHead()
{
        String wmlHead = "<?xml version=\"1.0\"?>\n" +


                 "<!DOCTYPE wml PUBLIC \"-//WAPFORUM//Dtd WML 1.2//EN\"" +
                 " \"http://www.wapforum.org/Dtd/wml_1.2.xml\">\n";
        return(wmlHead);
}
```

Methode: getAuftragTable

Liefert dem ShowAuftraege-Servlet eine Auftragstabelle mit Auftragauswahl und Auftragsbestätigungsbutton zurück.

```
public static String getAuftragTable()
        throws SQLException, ClassNotFoundException
{
        String query = "SELECT * FROM Auftrag WHERE fertig = 0;", name;
        int auftrag_nr, anzahl, tisch_nr, kellner_nr;
        StringBuffer auftrag = new StringBuffer(); // Erzeugen eines Stringbuffers
        ResultSet rs = getResultSet(query); // Datenbankabfrage

        auftrag.append("<body onload=\"Down()\">\n" + // Javascript für Aktualisierung
                       "<script language=\"JavaScript\"" +
                       "src=\"" + server + port1 +
"/webinterface/delaya.js\" type=\"text/javascript\"></script>\n");

        if(!((getResultSet(query)).next())) // Abfrage, ob Aufträge vorhanden
```

```
        {
                auftrag.append("<br>\n<h1> Keine Aufträge vorhanden! </h1>\n" +
                        "<h3> Freuen sie sich und machen Sie in der Zwischenzeit " +
                                "die Küche sauber." +
                                "</form>\n</body>\n</html>");
        }

        else    // Aufträge vorhanden → Ausgabe in HTML-Code
        {
                auftrag.append("<br><center><h3>Aktuelle Aufträge</h3><br>\n" +
                                "<form action=\"" + server + port2 +
"/servlet/AuftragFinished\" method=get>\n" +
                                "<table align=center border=1>\n" + "<tr bgcolor=red>\n" +
                                "<th>Markiert<th>Name<th>Anzahl<th>Tisch#\n");

                while(rs.next()) // Durchlauf durch gesamtes ResultSet
                {
                        auftrag_nr = rs.getInt("auftrag_nr");
                        name = rs.getString("name");
                        anzahl  = rs.getInt("anzahl");
                        tisch_nr = rs.getInt("tisch_nr");
                        auftrag.append("<tr><td align=center><input type=\"radio\"" +
                        "name=\"auftrag_nr\"" +
                                                " value=\"" + auftrag_nr + "\">" +
                                                "<td>"+ name +
                                                "<td align=center>"+ anzahl +
                                                "<td align=center>"+ tisch_nr + "\n");
                }
                auftrag.append("</table><br><br>\n");
                auftrag.append("<p>Markierter Auftrag fertig?</p>\n" + //Bestätigungsbutton
                                "<input type=\"submit\" name=\"Finished\" value=\"Löschen\">" +
                                "</form>\n</body>\n</html>");
        }
        return(auftrag.toString());   // Übergabe an Servlet
}
```

Methoden:

- getKellnerTable
- getSpeiseTable
- getSpeiseUpdateTable
- getUnbezahlt

Diese Funktionen werden aufgrund ihrer Ähnlichkeit zu getAuftragTable nicht näher erläutert. (nähere Beschreibung → Javadoc auf Projekt-CD)

Methode: updateDB

updateDB bekommt ein beliebiges SQL-Statement, stellt eine Verbindung zur MySQL-Datenbank her und führt das Update aus. Wird bei Dateneingabe, Datenmanipulation, Abrechnung und Auftragsbestätigung verwendet.

```
public static void updateDB(String update)
        throws SQLException, ClassNotFoundException
{
        Class.forName (driver);
        Connection con = DriverManager.getConnection(url,user,passwd); //Verbinden zur DB
        Statement stmt = con.createStatement();
        stmt.executeUpdate(update);
        con.close();   // Schließen der Verbindungen
        stmt.close();
}
```

Methode: calculate

Diese Funktion rechnet alle Aufträge eines Tisches zusammen, markiert anschließend die Aufträge als bezahlt und liefert den Gesamtbetrag zurück. Falls der Koch einmal auf die Auftragsbestätigung vergisst, wird hier der Auftrag auch gleichzeitig als fertig markiert.

```
public static int calculate(String tisch_nr)
        throws SQLException, ClassNotFoundException
{
        int betrag = 0;

        // Diese Query liefert Preis und Anzahl der Aufträge, die auf einer best. Tischnummer
           noch unbezahlt sind zurück.
String query = "SELECT preis, anzahl FROM Speisekarte, Auftrag WHERE" +
" Speisekarte.name = Auftrag.name AND (Auftrag.tisch_nr = " +
" tisch_nr + ") AND (Auftrag.bezahlt = 0);";

// Markiert die Aufträge als bezahlt
        String update1 = "UPDATE Auftrag SET fertig = 1 WHERE " +
                         "(tisch_nr = " + tisch_nr +") AND (bezahlt = 0);";

// Markiert die Aufträge als fertig
String update2 = "UPDATE Auftrag SET bezahlt = 1 WHERE " +
                         "(tisch_nr = " + tisch_nr +") AND (bezahlt = 0);";

        ResultSet rs = getResultSet(query);

        while(rs.next())      // Berechnung
        {
                betrag = betrag + (rs.getInt("preis")) * rs.getInt("anzahl");
        }
        updateDB(update1); // bezahlt
        updateDB(update2); // fertig
        return betrag;
}
```

Methode: checkKellner

Die Methode checkKellner bekommt Kellnernummer und Passwort von einer WML-Form, überprüft ob diese Werte mit der DB übereinstimmen und gibt true bzw. false zurück. Sie wird beim Kellnerlogin verwendet.

```
public static boolean checkKellner(String kellner_nr, String passwd)

        throws SQLException, ClassNotFoundException
{
        // Diese Query liefert ein Passwort bei geg. Kellnernummer zurück
String query = "SELECT passwd FROM Kellner WHERE kellner_nr =" + kellner_nr;
        ResultSet rs = getResultSet(query);

        if((rs.getString("passwd")).equals(passwd)) // Überprüfung
                return true;   // OK
        else
                return false;  // Fehler!
}
```

Methode: generateMenu

Diese Methode erstellt WML-Code für das WAP-Bestellmenü, in Abhängigkeit der aktuellen Speisetabelle. Wird immer dann gestartet, wenn der Kellner eine neue Bestellung aufnehmen will.

```
public static String generateMenu()
        throws SQLException, ClassNotFoundException
{
        StringBuffer menu = new StringBuffer(); // Erzeugen eines Stringbuffers (hinzufügen möglich)

        // Querries für Getränke, Vorspeisen, Hauptspeisen und Nachspeisen
String selectGtr = "SELECT name FROM Speisekarte WHERE art=\"gtr\";";
        String selectVsp = "SELECT name FROM Speisekarte WHERE art=\"vsp\";";
```

```
String selectHsp = "SELECT name FROM Speisekarte WHERE art=\"hsp\";";
String selectNsp = "SELECT name FROM Speisekarte WHERE art=\"nsp\";";

ResultSet rsGtr, rsVsp, rsHsp, rsNsp;
rsGtr = getResultSet(selectGtr);
rsVsp = getResultSet(selectVsp);  // Datenbankabfragen
rsHsp = getResultSet(selectHsp);
rsNsp = getResultSet(selectNsp);

menu.append(WMLHead());  // Start des WML-Codes
menu.append("<wml>\n" +
        "<head>\n" +
        "<meta http-equiv=\"Cache-Control\" content=\"no-cache\" forua=\"true\"/>\n" +
        "<meta http-equiv=\"Cache-Control\" content=\"no-store\" forua=\"true\"/>\n" +
        "</head>\n" +
        "       <card id=\"Bestellung\" title=\"WAPSCAS-Bestellung\">\n" +
        "       <p><small>Bestellung beginnen</small></p>\n" +
        " <p>\n" +
        "       <fieldset title=\"Tischnummer\">\n" +
        "        Tisch_nr:\n" +
        "               <input name=\"tisch_nr\" title=\"tisch_nr\" emptyok=\"false\"" +
                                    " format=\"N\"/>\n" +
        "       </fieldset>\n" +
        "       <fieldset title=\"Speisekarte\">\n" +
        "  <select multiple=\"false\" name=\"name\">\n");

//Getraenke
menu.append("   <optgroup title=\"Getraenk\">\n");
while(rsGtr.next())
{
        menu.append("    <option value=\"" + rsGtr.getString("name") + "\">" +
                        rsGtr.getString("name") + "</option>\n");
}

//Vorspeise
menu.append("   </optgroup>\n" +
        "   <optgroup title=\"Vorspeise\">\n");
while(rsVsp.next())
{
        menu.append("    <option value=\"" + rsVsp.getString("name") + "\">" +
                        rsVsp.getString("name") + "</option>\n");
}


//Hauptspeise
menu.append("   </optgroup>\n" +
        "   <optgroup title=\"Hauptspeise\">\n");
while(rsHsp.next())
{
        menu.append("    <option value=\"" + rsHsp.getString("name") + "\">" +
                rsHsp.getString("name") + "</option>\n");
}

//Nachspeise
menu.append("   </optgroup>\n" +
        "   <optgroup title=\"Nachspeise\">\n");

while(rsNsp.next())
{
        menu.append("    <option value=\"" + rsNsp.getString("name") + "\">" +
                rsNsp.getString("name") + "</option>\n");
}
menu.append("   </optgroup>\n");
menu.append("  </select>\n" +
        " </fieldset>\n" +
        " <fieldset title=\"Anzahl\">\n" +
        "  Anzahl:\n" +
        "  <input name=\"anzahl\" title=\"anzahl\" emptyok=\"false\"" +
              " format=\"N\"/>\n" +
        " </fieldset><br/>\n" +
        " <anchor>\n" +
        "  Senden\n" +
        "  <go method=\"get\" href=\"" + server + port2 + "/servlet/NewAuftrag\">\n" +
        "   <postfield name=\"tisch_nr\" value=\"$(tisch_nr)\"/>\n" +
        "   <postfield name=\"name\" value=\"$(name)\"/>\n" +
        "   <postfield name=\"anzahl\" value=\"$(anzahl)\"/>\n" +
        "  </go>\n" +
        " </anchor>\n<br/>" +
        "<a href=\"" + server + port1 + "/wml/main.wml\">Zurueck</a>\n" +
        "</p>\n" +
        "</card>\n" +
        "</wml>\n");

return(menu.toString()); // Stingbuffer in String umwandeln und an das Servlet übergeben
```

```
        }
}
```

4.2.2 Servlets

Die folgenden Servlets werden vom Webinterface aus gestartet und dienen zur Datenein – bzw. Datenausgabe. Da alle Servlets vom Aufbau her ziemlich ähnlich sind und sich nur die Namen der Funktionsaufrufe ändern, werden folgend nur 2 Servlets näher erläutert. (nähere Beschreibung → Javadoc auf Projekt-CD)

Servlet: NewKellner

Dieses Servlet bekommt Kellnerdaten von einer HTML-Form, schreibt diese Daten in die DB, gibt eine Bestätigungsmeldung aus und leitet zur Kellnerausgabe weiter.

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class NewKellner extends HttpServlet
{
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException
        {

                response.setContentType("text/html");
                PrintWriter out = response.getWriter();

                String insert = "INSERT INTO Kellner(vorname, nachname, passwd) " +
                                "VALUES( '" + request.getParameter("vorname") + "'," +
                                "'" + request.getParameter("nachname")+ "'," +
                                "'" + request.getParameter("passwd") + "');";



        try {
                        Wapscas.updateDB(insert);
                        out.println(Wapscas.headWithTitle("Kellnereingabe"));
                        out.println("<body onload=\"Down()\">");
                        out.println("<script language=\"JavaScript\"" +
"src=\"" + Wapscas.server + Wapscas.port1 + "/webinterface/delayk.js\"" +
                                "type=\"text/javascript\"></script>" +
"<h3>Ihr neuer Kellner wurde erfolgreich" + "eingetragen!</h3>\n</body></html>");
                }
                catch (SQLException se) {

                        out.println("SQLException: " + se.getMessage());
                }

                catch (ClassNotFoundException cnfe) {
                        out.println("Fehler: " + cnfe.getMessage());
                }
        }
}
```

Servlet: ShowKellner

Dieses Servlet liefert die eingetragenen Kellnern mit einer Funktion zum Löschen zurück.

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ShowKellner extends HttpServlet
{
	public void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException
	{
		response.setContentType("text/html");
		PrintWriter out = response.getWriter(); // Ausgabe festlegen

		try {
			out.println(Wapscas.headWithTitle("Eingetragene Kellner"));
			out.println(Wapscas.getKellnerTable());
		}
		catch(SQLException se) {
			out.println("Ein Fehler ist aufgetreten:" + se.getMessage());
		}
	  catch(ClassNotFoundException cnfe) {
			out.println("Ein Fehler ist aufgetreten:" + cnfe.getMessage());
		}
	}
}
```

4.2.3 WML – Programmierung

Dieses Beispiel sollte die WAP Seitenprogrammierung darstellen.

Die Datei heisst *index.wml* und wird als erstes in unseren WAP Interface angezeigt.

```
<!-- created by Dietmar Hafner (http://212.186.44.92:9080/wml/) -->
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>
<!-- Willkommen Seite / 1. Card -->
<card id="Welcome" title="WAPSCAS-Interface">
	<p align="center">
		<img src="Logo.wbmp" alt=" kein Logo"/><!--Einbinden eines WBMP Files-->
		<br/>
			Willkommen bei WAPSCAS!
			<br/>
			<a href="#Login">Weiter</a><!--Link erstellen-->
		</p>
</card>
<!-- 2.Card-->
<card id="Login" title="WASCAS-Login">
 <p align="left">
      <fieldset title="Kellnerdaten"><!--Felder gruppieren-->
      Kellnernr.:
      <input name="kellner_nr" title="Kellnernummer" format="N"/><!--Eingabefeld-->
      Passwort:
      <input name="passwd" title="Kellnerpasswort" type="password"/><!--Eingabefeld-->
      </fieldset>
    </p>

	<p><anchor><!--Ankerobjekt der als Link dienen sollte-->
			Login
<go method="get"href="http://212.186.44.92:8080/servlet/KellnerLogin">
				<postfield name="kellner_nr" value="$(kellner_nr)"/>
				<postfield name="passwd" value="$(passwd)"/>
			</go>
	</anchor>
	</p>
</card>
</wml>
```

4.3 Dateiformate

Die folgenden Dateiformate werden bei unseren Projekt eingesetzt.

*.css... Cascading Stylesheet File: Findet Anwendung beim Webinterface, einheitliches Layout.

*.class... Kompilierte Java Source File : Wird im Browser oder in der Konsole ausgeführt.

*.conf... Konfigurations File: Dient zur Konfiguration des Apache Webservers. Einfaches Textfile

*.html... Hyper Text Markup Language File: Ist eine Datei, die im Webbrowser angezeigt wird – unser Webinterface.

*.java... Java Source File : Enthält den eigentlichen Java Source Code.

*.jpeg, *.gif... Graphikdateien: Graphikelemente im Webinterface (größtenteils Gifs)

*.js... JavaScript File: Enthält ein Java Script, welches im Webinterface ausgeführt wird.

*.wml... Wireless Markup Language File: Ist eine Datei, welche am WAP Handy angezeigt werden kann – unser WAP Interface.

*.wbmp... Wireless Bitmap: Graphikformat welches am WAP Handy angezeigt werden kann.

4.4 Testdokumentation

Um unser Projekt testen zu können bedarf es einiger Einrichtungen wie Internetserver, Client PC und einem WAP Handy. Da diese Einrichtungen an der Schule nur teilweise vorhanden sind, mussten wir eine andere Lösung finden unser Projekt testen zu können.

Der Serverbetrieb wurde dann auf den PC von Herrn Hafner umgelegt. Dies hatte zur Folge, das bei der Programmierung unseres Projektkernstücks, den Java Servlets, zwischen zwei unterschiedlichen IP-Adressen unterschieden werden musste (PC in der Schule – PC zu Hause).

Die Funktionalität der Servlet konnte in den Unterrichtsstunden probiert werden, was wieder ein großer Vorteil war. Konfigurationen der Serverprogramme wurden auch in der Schule entwickelt und am PC, der zu Hause stand, angepasst. Die eigentlich Testumgebung unseres Projektes, belief sich auf drei Betriebssysteme: Microsoft Windows 98, 2000 Advanced Server und SuSE Linux. Aufgrund der „Plattformunabhängigkeit“ der Java Dateien kam es zu keinen erwähnenswerten Problemen.

Da auch WAP mit im Spiel war, wurden die Seiten zunächst nur auf einen WAP Simulator betrachtet. In weiterer Folge war dieses Testen nicht mehr möglich, da generierte Seiten von Simulatoren nicht richtig dargestellt werden konnten. Somit musste zum „wirklichen“ WAP Handy gegriffen werden. Dies erschwerte die Arbeit etwas, die im ersten Moment nie mit Bestimmtheit gesagt, werden konnte ob die generierte WAP Seite auch fehlerfrei ist.

Die Testfälle wurden nun die meiste Zeit online durchgeführt. Das heisst, die Server / Client Kommunikation sowie die Miteinbeziehung des WAP Handys lief nur mit einer Onlineverbindung. Das Testen wurden dann zu einem erweiterten Debugging umgewandelt, doch man „versteckte Fehler“ nur durch vieles und unterschiedliches Testen entdecken kann.

Wenn nun Fehler auftraten, so wurde sofort an deren Behebung gearbeitet (Bugs beseitigen).

4.5 Wartung

Bestellungen, die einmal aufgenommen wurden, werden nicht aus der Datenbank entfernt, wenn sie fertig oder bezahlt sind. Sie werden nur als fertig bzw. bezahlt markiert und scheinen daher nirgends mehr auf. Damit die Auftragstabelle nicht immer größer wird wurde eine kleine Wartungsfunktion geschrieben, die vom Webinterface aus gestartet werden kann.

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class AuftragReset extends HttpServlet
{
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException
        {
                String reset = "DELETE FROM Auftrag;"; // löscht alle Einträge

                response.setContentType("text/html");
                PrintWriter out = response.getWriter();

                try {
                        Wapscas.updateDB(reset);
                        out.println(Wapscas.headWithTitle("Aktuelle Aufträge"));
                        out.println("<br><h1>Die Auftragstabelle wurde zurückgesetzt!</h1>\n" +
                                    "</body></html>");
                }
                catch(SQLException se) {
                        out.println("Ein Fehler ist aufgetreten:" + se.getMessage());
                }
          catch(ClassNotFoundException cnfe) {
                        out.println("Ein Fehler ist aufgetreten:" + cnfe.getMessage());
                }
        }

}
```

5 Benutzerdokumentation

Die Benutzerdokumentation sollte dazu dienen, dem User möglichst einfach zu erklären, wie unser System zu installieren bzw. wie es in Betrieb genommen werden kann.

Hardware- und Softwareressourcen:

Server PC (Windows oder Unix PC, Onlineverbindung mit statischer IP Adresse, genügend Prozessorleistung für JAVA, unsere Softwarepakete)

Browser PC (Windows oder Unix PC, ebenfalls eine Onlineverbindung, Webbrowser)

WAP Handy (GSM Mobiltelefon, welches WAP unterstützt).

5.1 Installation und Konfiguration (für Win32 Systeme)

Der erste Schritt ist, unsere verwendeten Softwarepakete am Server PC zu installieren. Dazu gehören, der Apache Webserver (Installationshinweise befolgen), der Tomcat Server (siehe mitgelieferte Readme Datei), MySQL Datenbank (per Setup) und Java Developer Kit (per Setup).

Wenn am Browser PC kein Internet Browser installiert ist, so ist dies nachzuholen (z.B.:MS Internet Explorer 5.5).

Server und JDK- Pfade setzen:

Setzen des Tomcat Home Pfades: in der Autoexec.bat (bei Win2k Umgebungsvariablen) die Zeile

SET TOMCAT_HOME=C:\tomcat

Setzen des JDK Pfades: ebenfalls in der Autoexec.bat die Zeile

SET PATH=C:\JDK1.3\bin

Setzen des Pfades für die Datei Servlet.jar (wird benötigt für JDK, um Servlets kompilieren zu können)

SET CLASSPATH=C:\tomcat\web-apps\ROOT\web-inf\classes

Apache Server Konfiguration:

Das mitgelieferte *httpd.conf* File muss nach C:\Programme\Apache Group\Apache\conf kopiert werden. (Achtung dieses File kann nicht für Unix Systeme verwendet werden, da die Pfadesangaben nicht stimmen)

Da man nun das Webinterface nach außen hin schützen will, wird der Zugriff per Login realisiert. Es müsse somit Benutzer mit Passwörtern erstellt werden. Dazu geht man folgend vor:

Eine MS-DOS Box wird geöffnet, ins *C: \Programme\Apache Group\Apache\bin* Verzeichnis muss gewechselt werden. Danach wird „ *htpasswd –c passwords Benutzername*“ eingegeben (Benutzername kann frei gewählt werden). Als nächstes verlangt die Anwendung nach einem Passwort, welches hiermit vergeben werden kann. Des wars auch schon mit dem Einrichten von Benutzern. Dieser Prozess kann auf beliebig viele Benutzer angewandt werden. Zu beachten ist dabei, dass sich nur der Benutzername im Konsolenbefehl ändert.

Kopieren des Systemkernstücks:

Der Ordner **Webinterface** muss nach *C:\Programme\Apache Group\Apache\htdocs* gezogen werden.

Der Ordner **wml** muss nach *C:\Programme\Apache Group\Apache\htdocs* gezogen werden.

Die **Dateien** im Ordner Java Servlets muss nach *C:\tomcat\web-apps\ROOT\web-inf\classes* gezogen werden.

Starten der Anwendungen:

- Tomcat Server mit der Verknüpfung startup.bat starten
- Apache Server mit ebenfalls mit der Verknüpfung starten
- MySQL per Verknüpfung starten

Ausführungen vor dem Systemstart:

CreateTables in der Konsole folgend ausführen: eine MS-DOS-Eingabeaufforderung öffnen, in den Pfad der installierten Java Servlets wechseln (*C:\tomcat\web-apps\ROOT\web-inf\classes*)

und „ java *CreateTables*“ eingeben.

Die Datei **Wapscas.java** mit einen Texteditor öffnen und die Variable Server auf die vom Internetprovider zugewiesene IP-Adresse ändern (MS-DOS-Eingabeaufforderung öffnen und winipcfg eingeben, unter Win2k MS-DOS Box öffnen und ipconfig eingeben). Der Wert für den Port muss nicht verändert werden, außer es befindet sich bereits an andere Anwendung am jeweiligen Port (9080 für Apache und 8080 für Tomcat-Server).

Danach ist die Datei **Wapscas.java** mit dem Konsolenbefehl „*javac Wapscas.java*“ zu kompilieren.

5.2 Bedienungsanleitung für das Webinterface

Nach Befolgung der oberen Punkte sollte unser System bereits einsatzfähig sein.

Nun muss folgend vorgegangen werden, um das System nutzen zu können.

Zu aller erst müssen ein paar Daten per Webinterface eingegeben werden. Dies kann entweder am Server oder am Browser PC erfolgen.

Ein Internet Explorer Fenster öffnen – Die Adresse http://localhost:9080/webinterface oder http://IP-Adresse des Servers:9080/webinterface eingeben.

Der folgende Bildschirm erscheint.

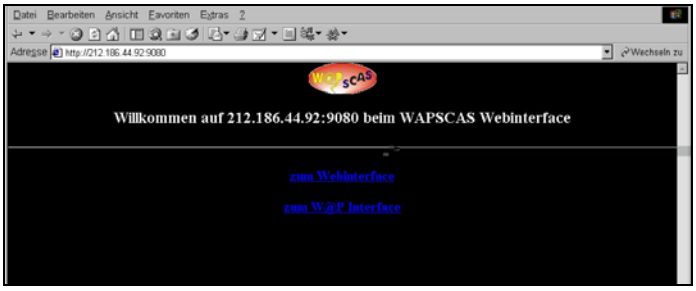

Abbildung 12

Nun muss auf „*zum Webinterface*“ geklickt werden. Darauf hin kommt ein Loginfenster zum Vorschein

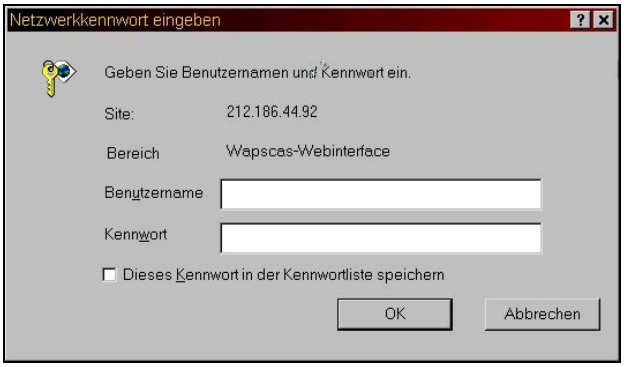

Abbildung 13

Nun muss der Benutzername und das Kennwort eingegeben werden. Zur Einrichtung individueller Benutzernamen und Passwort siehe Konfigurieren des Apache Webservers.

Daraufhin erscheint das Webinterface, welches nun einsatzbereit ist.

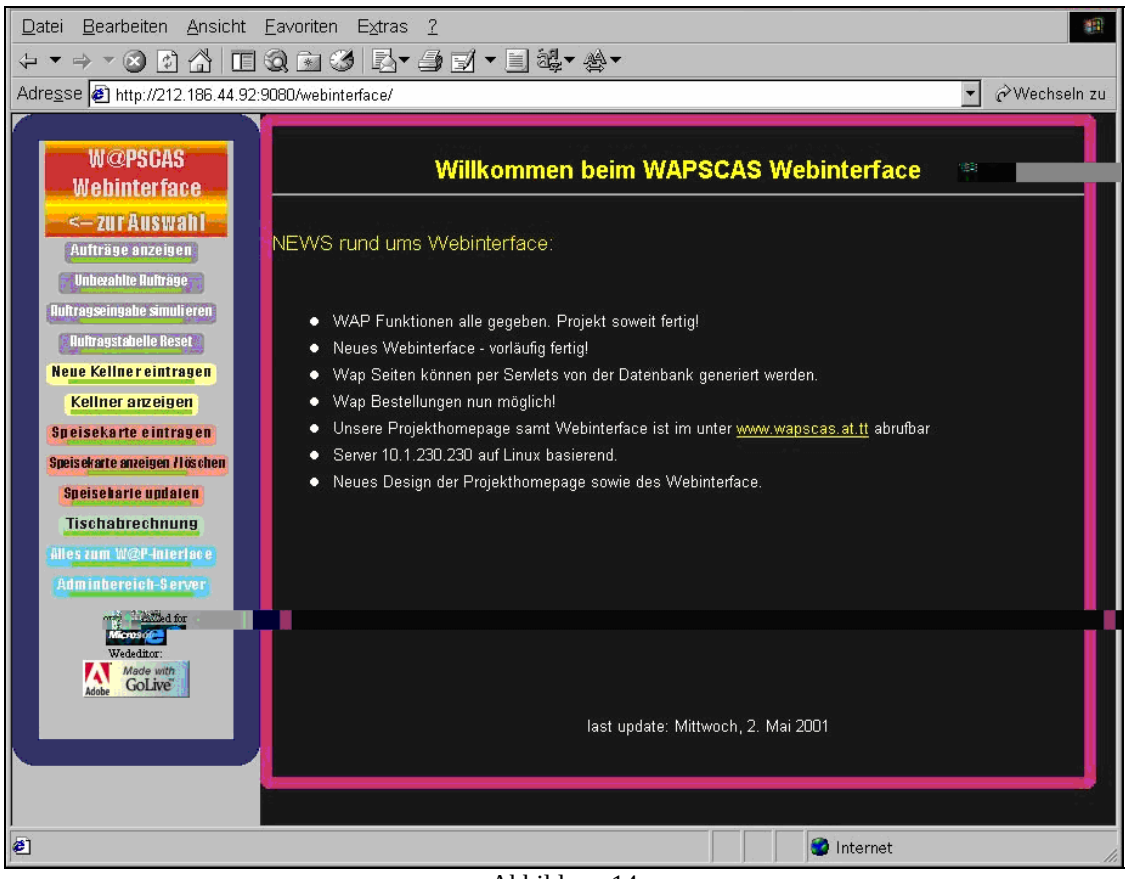

Abbildung 14

Das Webinterface ist soweit fertig eingerichtet doch man will natürlich auch das WAP Handy nutzen. Um dies tun zu können muss noch folgende Punkte befolgt werden:

Da man sich am WAP Interface ebenfalls einloggen muss, sind Kellner einzutragen.

Kellnervor- und Nachname und ein Passwort muss dazu vergeben werden. Zu Beachten ist, dass das Passwort bestätigt werden muss (zweimal richtig eingegeben).

Wichtig: Eine Kellnernummer wird automatisch erstellt. Diese dient als Benutzerkennung am WAP Interface. Also Kellner müssen sich ihre Kellnernummer und ihr Passwort unbedingt merken.

Es muss auch ein jedes Eingabefeld etwas eingetragen werden!

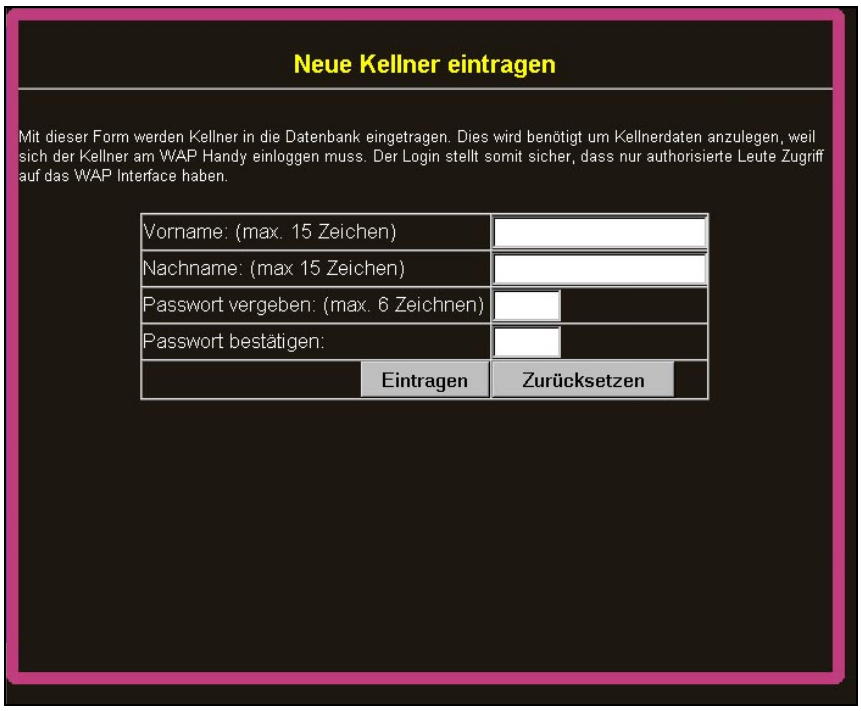

Abbildung 15

Nach Klicken auf den Eintragenbutton wird der neue Kellner in die Datenbank eingetragen. Eine Bestätigung erscheint am Bildschirm. Nach 5 Sekunden wird zur Kellnerliste weitergeleitet, die alle eingetragenen Kellner anzeigt. In dieser Liste (Menüpunkt „Kellner anzeigen“) können Kellner wieder aus der Datenbank entfernt werden.

Um am Handy auch etwas bestellen zu können müssen Speisen und Getränke in die Datenbank eingetragen werden. Dazu muss auf „*Speisekarte eintragen*“ im Menü geklickt werden. Es erscheint folgendes Bild.

Abbildung 16

Bei Name ist Name des Getraenks, der Vor-, der Haupt- oder der Nachspeise einzutragen. Die Art wird nun noch gewählt und der Preis wird festgelegt. Nach Klicken auf „*Eintragen*“ erscheint ebenfalls eine Bestätigung und wird man zur Speisekarte weitergeleitet.
Mit dem Menüpunkt „*Speisekarte anzeigen*“ werden erstens alle Speisen und Getränke angezeigt werden, sowie einzelne Dateneinträge können gelöscht werden. Mit „*Speisekarte updaten*“ kann dann noch der Preis verändert werden.

Wichtig: Da die Speisekarte, welche am WAP Handy angezeigt wird, immer mit den Daten der Datenbank generiert wird, muss es in jeder Kategorie (Getränk, Vor-, Haupt, und Nachspeise) ein Eintrag gemacht werden. Ansonsten kann am Handy nichts bestellt werden.

Nun kann am Handy schon bestellt werden, doch eine genauere Einleitung dazu gibt es später.

Nun wird noch das Anzeigen der Aufträge behandelt:

Mit dem Menüpunkt „*Aufträge anzeigen*“ werden die aktuellen Aufträge angezeigt. Die Seite wird alle fünf Sekunden aktualisiert und ist somit immer am neuersten Stand.

Es kann ein Auftrag markiert und dann gelöscht werden. Das signalisiert, das der Auftrag abgeschlossen ist.

Weiters gibt es einen Menüpunkt „*unbezahlte Aufträge*“. Dieser wird dann benötigt, wenn es Unklarheiten im Betrieb gibt, bei welchem Tisch schon abgerechnet wurde.

Wichtig: Diese Seite wird nicht automatisch aktualisiert!

Ein Notfallkonzept wurde am Webinterface ebenfalls entwickelt. Da auch GSM Netze ausfallen können, kann am Webinterface ebenfalls bestellt werden, sowie abgerechnet („*Tischabrechnung*“ und „*Auftragseingabe simulieren*“)

Mit dem Punkt „Auftragstabelle Reset“ können alle Aufträge, bezahlte wie auch unbezahlte, aus der Datenbank entfernt werden. Zur Sicherheit wird bei diesem Vorgang noch eine Bestätigung des Anwenders abgefragt.

Abbildung 17

5.3 Bedienungsanleitung für das WAP Interface

Nun ist es soweit, es kann zum wapfähigen Handy gegriffen werden. Es ist noch zu beachten, das die WAP Fähigkeit des Handys entsprechend dem GSM Anbieter konfiguriert ist.

Um nun ins WAP Interface zu gelangen, müssen folgende Punkte befolgt werden.

(Das Beispiel wird anhand eines Nokia Gerätes erklärt)

Am Handy ins **Hauptmenü** wechseln, **Dienste** wählen, dann **zu Adresse** und folgendes eingeben: http://212.186.44.92:9080/wml/index.wml/

Zu beachten ist dabei, das der „/“ am Ende nicht vergessen wird. Um nun nicht immer mühselig die gesamte Adresse eingeben zu müssen, kann die Adresse auch als Lesezeichen gespeichert werden.

Danach sollte nach ein paar Sekunden folgender Bildschirm erscheinen.

Abbildung 18

Es muss nun auf „*Weiter*“ geklickt werden (klicken in dem Sinn gibt es nicht, doch anwählen mit den Pfeilen am Handy und „*Optionen*“ „*Auswählen*“ drücken)

Danach erscheint z.B. Folgendes:

Abbildung 19

Kellnernummer und Kellnerpasswort müssen eingegeben werden, um fortfahren zu können.

Beim richtigen Logindaten wird man zum Hauptmenü des WAP Interfaces weitergeleitet, bei falschen Daten kommt man zum Anfang zurück.

Nun kann mit der „Kellnerei“ begonnen werden. „*Neue Bestellung*“ wird somit angewählt um Bestellungen aufnehmen zu können.

Der Bildschirm am Handy sieht dann z.B. so aus:

Abbildung 20

Bei einer neuen Bestellung muss die *Tischnummer* als erstes, die *Bestellung* an sich als zweites und die *Anzahl* als letztes eingegeben werden. Zur Bestellung bleibt zu sagen, dass die einzelnen Einträge der Speisekarte hierarchisch gegliedert sind (siehe linkes Bild). Danach kann „*Senden*“ oder „*Zurück*“ gewählt werden.

Abbildung 21

Abbildung 22

Zu guter letzt wird noch die Abrechnung am Handy beschrieben.

Dazu muss man sich im Hauptmenü befinden und „*Abrechnung*“ wählen.

Die Tischnummer muss eingegeben werden und danach „*Abrechnen*“ ausgewählt werden.

Nach ein paar Sekunden erscheint der Rechnungsbetrag am Handy.

Abbildung 23

5.4 Mögliche Fehlermeldungen (Web- und WAP Interface)

5.4.1 Webinterface

Da im Webinterface sehr viel eingegeben werden kann, treten hier am häufigsten Fehler auf.

Der erste serverseitige Fehler, der entstehen kann ist:

Die Seite wurde nicht gefunden.

Die gewünschte Seite wurde möglicherweise entfernt oder umbenannt, oder sie ist vorübergehend

Fehlerursache: Der Apache Server ist nicht gestartet, vertippt bei der Adresse oder momentan keine Onlineverbindung.

Fehlerbehebung: Server starten oder Webinterface auf einen anderen internetfähigen PC ausführen.

Beim Login ins Webinterface können ebenfalls Fehler auftreten. Der Benutzername oder das Passwort kann falsch sein.

Folgendes geschieht:

Fehlerursache: Passwort vergessen oder vergessen etwas eingegeben zu haben

Fehlerbehebung: Passwort richtig schreiben, notfalls Benutzereinrichtung wiederholen (siehe Apache Server Konfiguration)

Im Webinterface kann ebenfalls folgender Fehler auftreten, nachdem auf Aufträge anzeigen, unbezahlte Aufträge, Kellner anzeigen, Speisekarte anzeigen/löschen, Speisekarte updaten geklickt wurde.

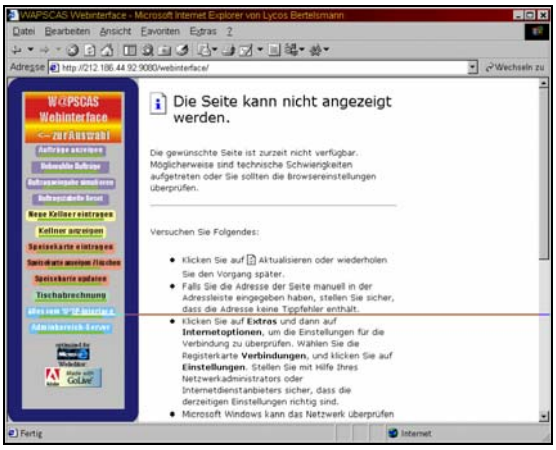

Abbildung 24

Fehlerursache: Tomcat Server ist nicht gestartet, Onlineverbindung ist in dem Moment ausgefallen.

Fehlerbehebung: Tomcat Server starten, Webinterface auf einen anderen internetfähigen PC ausführen.

Nun wird auf die clientseitigen Fehler eingegangen, die beim „Dateneintragen“ auftreten können. Die Fehlerabfragen wurden mit JavaScript realisiert.

Beginnen wir mit dem Kellner eintragen:

Es muss in jedes Eingabefeld etwas eingetragen werden (Vor-, Nachname, sowie 2 mal das Passwort) Die Länge wird von der HTML Seite her bestimmt. Ansonsten könnte es zu folgender Fehlermeldung kommen.

Abbildung 25

Bei der Speisekarte können dieselben Fehler auftreten. Auch hier müssen Daten eingetragen werden. Es ist zu beachten, dass der Preis natürlich eine Zahl sein muss.

Auch das wurde gelöst.

Abbildung 26

Bei „Speisekarte updaten“ könnte es auch einen Fehler geben, wenn man bei der Preisveränderung z.B. keine Zahl eingeben würde. Dieses Problem wird gleich dem Speisekartenpreisproblem gelöst.

Noch eine letzte Bestätigung, die erscheinen kann.

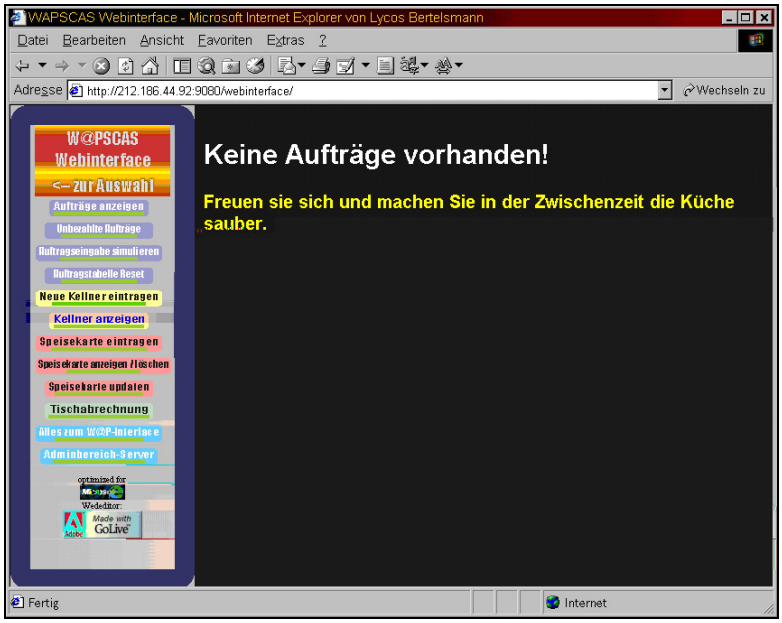

Abbildung 27

5.4.2 WAP Interface

Im WAP Interface können eigentlich nicht sehr viele Fehler auftreten. Der Login Fehler wurde bereits bei der Betriebsanleitung beschrieben.

Verbindungsfehler sind jedoch schon möglich.

Fehlerursache: Der Apache Server ist nicht gestartet, vertippt bei der Adresse oder momentaner GSM Netzausfall.

Fehlerbehebung: Server starten oder Notfallskonzept im Webinterface nutzen.

Abbildung 28

Bei Tomcat Server Fehlern kann folgendes am Handybildschirm erscheinen.

Abbildung 29

Fehlerbehebung: Tomcat Server neustarten.

6 Diskussion

6.1 Probleme bei der Projektentwicklung

WAP

Dadurch, dass unser Server nur im schulinternen Netzwerk und nicht nach außen hin sichtbar war, waren die WAP – Versuche nur mit einem lokalen Simulator (Nokia WAP-Toolkit) möglich. Nach einiger Zeit stellte sich aber heraus, dass dieser Simulator oft abstürzte und oft nicht das gewollte Ergebnis lieferte. Daher mussten wir einen privaten Server im Internet laufen lassen, um die WAP – Versuche direkt am WAP – Handy über das WAP – Gateway zu testen. Testläufe am Handy waren daher nur außerhalb der Unterrichtszeit möglich.

Getrennte Arbeitsplätze

Ein kleineres Problem waren die getrennten Arbeitsplätze. Dadurch, dass jeder von uns einen Wechselrahmen benötigte und nicht jeder Rechner einen Einschub dafür hatte, war es nötig getrennte Labors aufzusuchen. Eine Kommunikation während der Entwicklung war daher nur erschwert möglich.

LINUX

Einer unserer Rechner lief unter LINUX. Da dieses Betriebssystem etwas schwieriger zu konfigurieren ist, gab es öfters kleinere Probleme, die jedoch innerhalb kurzer Zeit von unserem Projektbetreuer gelöst wurden.

6.2 Erweiterungsmöglichkeiten

Kellner des Tages

Eine Erweiterung, die bereits gedanklich geplant war, ist die Funktion „Kellner des Tages“. Diese Funktion soll es ermöglichen, am Ende eines Tages, den Kellner mit den meisten Bestellungen zu

ermitteln. Dazu wären nur eine Erweiterung der Auftragtabelle und eine entsprechende Funktion notwendig gewesen.

Benutzerrechte für das Webinterface

Für das WAPSCAS – Webinterface gibt es zur Zeit nur einen Usernamen und ein Passwort d.h. Koch und Chef des Restaurants haben die gleichen Rechte. Dadurch könnte beispielsweise der Koch auch die Preise der Speisen ändern. Verschiedene Benutzerrechte könnten dies verhindern.

Tages- bzw. Jahresstatistik

Eine Funktion die alle Bestellungen eines Tages- bzw. Jahres auswertet und in einer Statistik auflistet könnte interessant und hilfreich für die Restaurantbesitzer sein.

Logistiksystem - Onlinebestellung

Einer Erweiterung der Speisetabelle oder die Einführung einer Bestandstabelle könnte es ermöglichen, dass WAPSCAS nach jedem Auftrag überprüft, ob noch genügend auf Lager ist und ggf. eine Online – Nachbestellung durchführt.

Mehrfachbestellung

Bei der aktuellen Bestellungsaufnahme ist es nur möglich eine Speise nach der anderen zu Bestellen. Eine Bestellung von mehreren Produkten gleichzeitig würde den Bestellvorgang etwas beschleunigen.

Unbezahlte und aktuelle Aufträge am Handy

Gegenwärtig kann der Kellner nur bestellen und abrechnen. Eine Funktion die unbezahlte und aktuelle Aufträge auch am Handy anzeigen kann, wäre hilfreich für Kellner und Kunde.

Rechnung ausdrucken

Wenn der Kunde eine Rechnung haben will, ist es zurzeit nur möglich, diese mit der Hand zu schreiben. Eine Druckfunktion könnte dem Kellner die Arbeit ersparen.

6.3 Vorschläge für weitere Entwicklung

Die zuvor beschriebenen Erweiterungsmöglichkeiten könnten mindestens ein Ganzjahresprojekt für 2 Personen füllen. Daher wären diese Erweiterungen ein perfektes Maturaprojekt für die folgenden Jahrgänge.

7 Literaturverzeichnis

Es werden hier nur verwendete Bücher angeführt. Die entsprechenden Internetseiten, wo wir uns auch sehr viel Informationen holten, sind nicht enthalten.

Name	Author	Verlag	Jahr	ISBN
MySQL&mSQL	Randy Jay Yarger, George Reese, Tim King	O'REILLY	1999	1-56592-434-7
Datenbanken&Java	Gunter Saake, Kai-Uwe Sattler	dpunkt	2000	3-932588-54-1
Servlets and JavaServer Pages	Marty Hall	Sun	2000	0-13-08934904
Java Programmierung mit IBM Visual Age	Florian Hawlitzek	Addison-Wesley Longman	1999	3-8273-1526-3
WML - die wireless markup language	Felix Brosius	Addison-Wesley	2000	3-8273-1695-2

Viel Spaß mit W@PSCAS wünscht das Projektteam!

Anhang A:

Grundlagen zur Diplomarbeit

Betriebssichere Client/Serverkopplung

Bearbeiter: Antonitsch Raimund

Inhaltsverzeichnis:

1 Aufgabenstellung 3
1.1 Allgemeines 3
1.2 Pflichtenheft 3
2 Theoretische Erörterung 3
2.1 Allgemeines zum Client/Server – Modell 3
2.2 Wahl der Server 4
2.2.1 Direkte Überprüfung auf Funktion 4
2.2.2 Indirekte Überprüfung mit Master-Server 5
2.2.3 Manuelle direkte Überprüfung 6
2.3 Redundanz der Server 6
2.3.1 Applikationsgestützt 6
2.3.2 Datenbankseitig 7
3 Praktische Umsetzung 7
3.1 Allgemeines 7
3.1.1 Java AWT (Abstract Window Toolkit) 7
3.1.2 Replikation in MySQL 8
3.2 Grafische Java-Applikation 8
3.3 CheckServer-Methode in Wapscas Hauptklasse 11
3.4 Serverüberprüfung am Handy 11
3.5 MySQL-Konfiguration 12
4 Ergebnisse 13
4.1 Java-Applikation 13
4.2 Server-Redundanz 15
4.2.1 Testapplikation 16
4.2.2 Kommunikationsaufwand 18
5 Schlussfolgerung 19

1 Aufgabenstellung

1.1 Allgemeines

Eine der am häufigsten gestellten Fragen über WAPSCAS war: Was passiert, wenn der Server ausfällt oder die Verbindung zu ihm nicht mehr besteht? – Die Antwort darauf war, dass bei Auftreten dieses Falles das ganze System lahmgelegt ist und keine Bestellungen mehr entgegengenommen werden bzw. keine Abrechnung mehr erfolgen könnte (zumindest nicht auf digitalem Wege).

1.2 Pflichtenheft

Es sollte ein theoretisches Modell erstellt werden, wie eine Client/Serverkopplung im Internet ausfallsicher gemacht werden kann. Für die Realisierung ist es notwendig, ein eigenes Protokoll zwischen redundanten Servern zu entwerfen. Der dabei entstandene Kommunikations-Overhead sollte minimiert werden. Das entworfene Modell sollte anhand eines Prototypen realisiert und der Kommunikationsaufwand mithilfe einer Protokollanalyse ermittelt werden.

2 Theoretische Erörterung

2.1 Allgemeines zum Client/Server – Modell

Strukturieren der Anwendungsschicht (OSI-Schicht 7)

- *Rollen:*
  - Dienstnachfrager (Client)
  - Diensterbringer (Server)

- *Rollenspezifisches Kommunikationsverhalten:*
  - Client: Anfrage senden, ev. Antwort abwarten
  - Server: Anfrage entgegennehmen, Anfrage bearbeiten, ev. Antwort senden

Vorzüge (gegenüber einer monolithischen/nicht – verteilten Architektur)

- *Ressourcen – Sharing*
  Teure Ressourcen können einer großen Anzahl von Benutzern auch über größere Distanzen hinweg angeboten werden.

- *Erhöhte Ausfallsicherheit*
  Durch Redundanz der Server (z.B. Filereplikation) kann die Zuverlässigkeit des Systems erhöht werden.

- *Kostenersparnisse*
  Kleinrechner haben ein besseres Preis/Leistungsverhältnis als Mainframes

- *Skalierbarkeit*
  Wachsende Leistungsbedürfnisse können durch sukzessives Hinzufügen einzelner Server befriedigt werden (statt Totalersatz der zentralen Lösung).

2.2 Wahl der Server

In den folgenden 3 Beispielen möchte ich Möglichkeiten für die Serverwahl näher beschreiben und Vorteile, sowie Nachteile der jeweiligen Methode näher erläutern.

2.2.1 Direkte Überprüfung auf Funktion

Am Küchen – PC (Client) ist eine Applikation gestartet, die überprüft, welcher Server funktionsfähig ist und automatisch eine Verbindung zum jeweiligen Server herstellt.

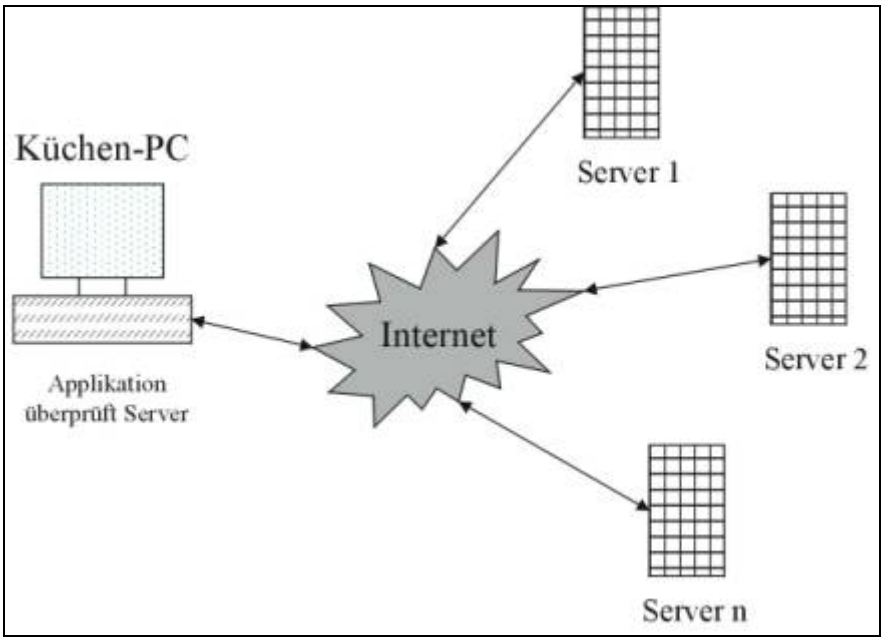

Abbildung 1

Realisierungsmöglichkeiten:

- **Java – Applet im Webinterface**

  Ein Java – Applet könnte eine Überprüfung der Server durchführen und bei einer evtl. Serverfehlfunktion den Server automatisch wechseln. Einziges Problem dabei ist jedoch, dass es lt. Sun einem Applet nicht erlaubt ist, mit einem anderen Host, außer dem Host, von dem das Applet geladen wurde, zu kommunizieren.

- **Applikation mit grafischer Oberfläche**

  Anstatt des Webinterfaces, welches im Browser geöffnet werden kann, könnte beispielsweise eine Java-Applikation die selben Funktionen wie das Webinterface

aufweisen und zusätzlich noch die Serverüberprüfung durchführen. Dadurch wäre es aber nicht möglich, von jedem beliebigen PC aus WAPSCAS zu verwenden. Eine Installation des Programms wäre zuerst erforderlich.

Beide Methoden können in dieser Weise am Handy nicht realisiert werden, da WML-Script keine Funktionen dafür zur Verfügung stellt.

2.2.2 Indirekte Überprüfung mit Master-Server

Der Küchen-PC kommuniziert mit einem Master-Server. Dieser Server überprüft die anderen Server und leitet je nachdem welcher aktiv bzw. bevorzugt wird weiter. Vorraussetzung dafür ist jedoch die ständige Bereitschaft (Ausfallssicherheit 100%) des Master-Servers. Fällt der Master-Server aus, steht das ganze System.

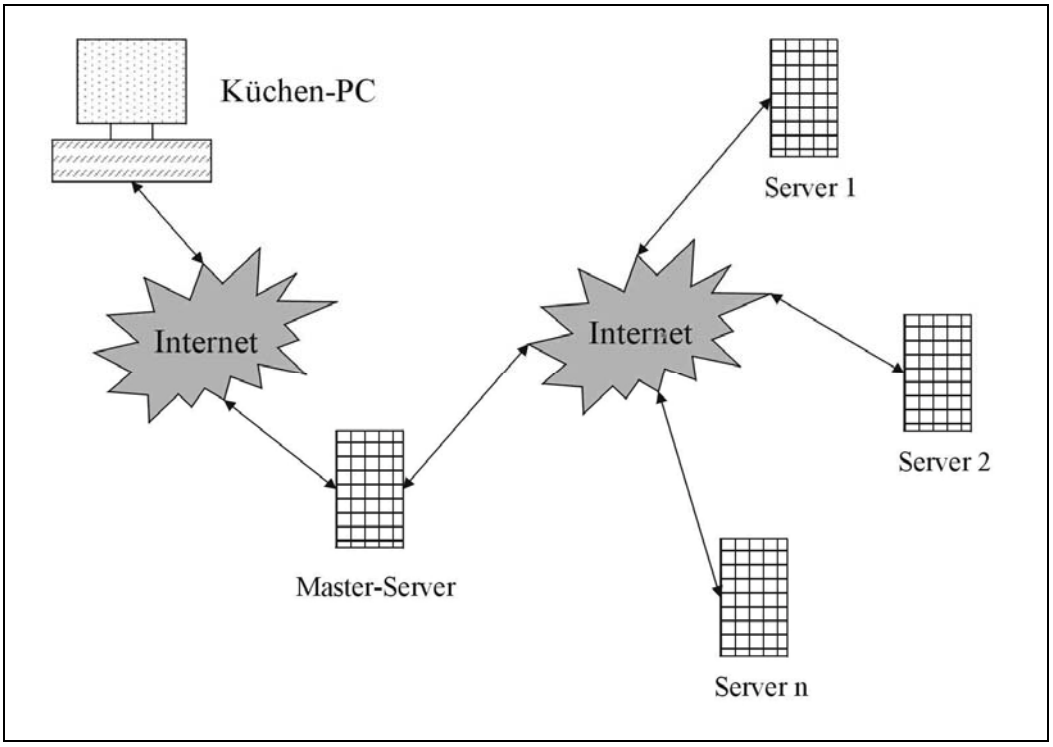

Abbildung 2

Realisierungsmöglichkeiten:

- **Applikation am Master-Server**

  Ein Java-Programm könnte bei jeder Client-Anfrage die Server prüfen und anschließend die Anfrage weiterleiten. Die Funktionsweise ist ähnlich wie die zuvor erwähnte Möglichkeit (direkte Überprüfung) nur mit dem Unterschied, dass die Applikation keine grafische Oberfläche benötigt und nur zur Serverprüfung verwendet wird.

- **Servlet am Master-Server**

  Da WAPSCAS Servlets verwendet, wäre es naheliegend, ein Servlet für die Serverwahl du verwenden. Dieses könnte direkt vom Webinterface aus gestartet werden und müsste nicht, wie bei der 1. Variante, über Sockets mit dem Master-Server kommunizieren.

2.2.3 Manuelle direkte Überprüfung

Die am einfachsten zu realisierende Methode ist die manuelle direkte Serverprüfung. Eine Java-Applikation wird am Küchen-PC (Client) gestartet, überprüft die Server und zeigt anschließend die aktiven Server an. Mit dieser Information kann der Benutzer mit Hilfe von Bookmarks den Server auswählen.

Am Handy findet die Überprüfung mit Hilfe eines Servlets statt. Der Kellner kann mit Bookmarks den Server wählen und bekommt je nachdem, ob der Server aktiv ist oder nicht eine Bestätigungs- bzw. Fehlermeldung zurück.

2.3 Redundanz der Server

Um die Redundanz der Server zu gewährleisten, werden nun folgende 2 Möglichkeiten dafür näher beschrieben:

2.3.1 Applikationsgestützt

- Jeder DB-Server hat eigene Applikation
  2 oder mehrere Applikationen auf jeweils einem Rechner kommunizieren miteinander und sorgen für Redundanz.

- Applikation auf Master-Server
  Eine Applikation auf nur einem Rechner (Master-Server) kommuniziert mit 2 oder mehreren Datenbanken und gewährleistet Redundanz. Fällt dieser Server aus, gibt es keine Redundanz.

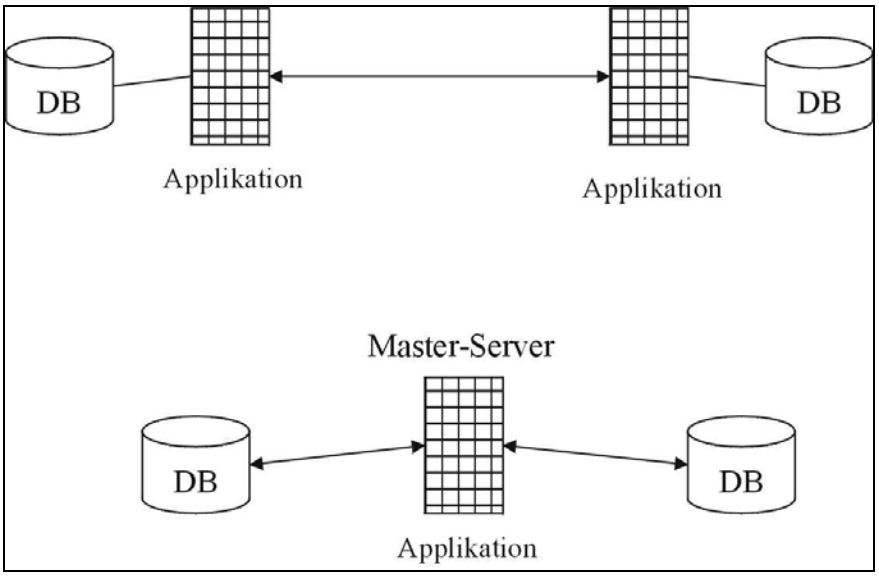

Abbildung 3

2.3.2 Datenbankseitig

Eine bereits integrierte Funktionalität der Datenbank sorgt dafür, dass 2 oder mehrere Datenbanken die gleichen Daten enthalten.

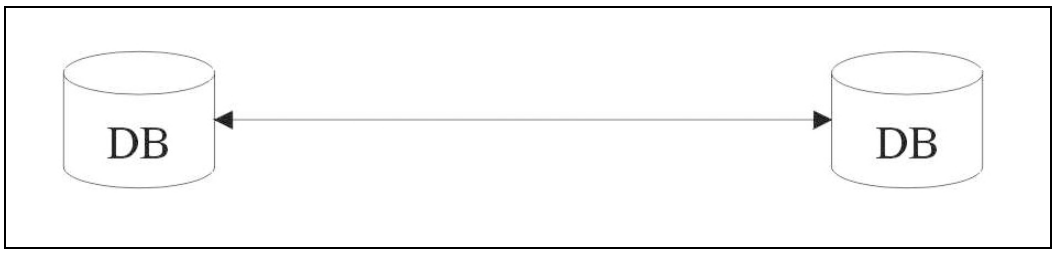

Abbildung 4

3 Praktische Umsetzung

3.1 Allgemeines

3.1.1 Java AWT (Abstract Window Toolkit)

Im Gegensatz zu den meisten anderen Programmiersprachen wurde Java von Anfang an mit dem Anspruch entwickelt, ein vielseitiges, aber einfach zu bedienendes System für die Gestaltung grafischer Oberflächen zur Verfügung zu stellen. Das Resultat dieser Bemühungen steht seit dem JDK 1.0 als Grafikbibliothek unter dem Namen *Abstract Windowing Toolkit* (*AWT*) zur Verfügung.

Die Fähigkeiten des AWT lassen sich grob in vier Gruppen unterteilen:

- Grafische Primitivoperationen zum Zeichnen von Linien oder Füllen von Flächen und zur Ausgabe von Text

- Methoden zur Steuerung des Programmablaufs auf der Basis von Nachrichten für Tastatur-, Maus- und Fensterereignisse
- Dialogelemente zur Kommunikation mit dem Anwender und Funktionen zum portablen Design von Dialogboxen
- Fortgeschrittenere Grafikfunktionen zur Darstellung und Manipulation von Bitmaps und zur Ausgabe von Sound

Da die grafischen Fähigkeiten Bestandteil der Sprache bzw. ihrer Klassenbibliothek sind, können sie als portabel angesehen werden. Unabhängig von der Zielplattform wird ein GUI-basiertes Programm auf allen verwendeten Systemen gleich oder zumindest ähnlich laufen.

3.1.2 Replikation in MySQL

Seit Version 3.23.15 unterstützt MySQL Replikation intern. Ein Server ist der Master, während ein anderer der Slave ist. Der Master schreibt alle Änderungen (Updates, Deletes, etc.), die an der Datenbank vorgenommen werden, in ein sog. „binary log“. Wenn der Slave zum Master verbindet, überprüft der Slave, welche Updates seit seiner Abwesenheit durchgeführt wurden und führt sie anschließend selbst aus.

3.2 Grafische Java-Applikation

Das folgende Programm dient zur Überprüfung der Verfügbarkeit von 3 Servern. Mit Hilfe des AWT (Abstract Windows Toolkit) von Java wird zunächst eine grafische Oberfläche aufgebaut.

Diese grafische Oberfläche enthält die IP-Adressen der zu überprüfenden Server und 2 Buttons. Einer zum Starten der Überprüfung und einer zum Beenden des Programmes.

Die Methode actionPerformed reagiert auf den Check- bzw. Beenden-Button und beendet das Programm, oder startet die Überprüfung.

```
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.net.*;

public class ServerCheck
extends Frame
implements ActionListener
{
  String driver = "org.gjt.mm.mysql.Driver"; //MySQL-JDBC-Treiber
  String server1url = "jdbc:mysql://10.1.230.230/WAPSCAS";
  String server2url = "jdbc:mysql://10.1.230.231/WAPSCAS";
  String server3url = "jdbc:mysql://10.1.11.105/WAPSCAS";
  String server1user = "root";      //MySQL-Benutzername
  String server2user = "root";
  String server3user = "root";
  String server1passwd = "";  //MySQL-Passwort
  String server2passwd = "";
```

```
  String server3passwd = "";
  TextField server1, server2, server3;
  Label server1aktiv, server2aktiv, server3aktiv, checkStatus;

  public static void main(String[] args)
  {
    ServerCheck wnd = new ServerCheck();  // Erzeugen eines neuen Fensters
    wnd.setVisible(true);                 // Fenster darstellen
  }


  public ServerCheck()
  {
      super("WAPSCAS - Servercheck");     // Fenster mit Titel
      addWindowListener(new WindowClosingAdapter(true));
      setBackground(Color.orange);
      setSize(400,350);
      setLayout(null);  // null-Layout zuweisen
      server1 = new TextField("10.1.230.230");
      server2 = new TextField("10.1.230.231");
      server3 = new TextField("10.1.11.105");
      server1.setBounds(130, 100, 100, 20);
      server2.setBounds(130, 150, 100, 20);
      server3.setBounds(130, 200, 100, 20);
      server1aktiv = new Label("aktiv");
      server1aktiv.setBounds(260,110,50,10);
      server2aktiv = new Label("aktiv");
      server2aktiv.setBounds(260,160,50,10);
      server3aktiv = new Label("aktiv");
      server3aktiv.setBounds(260,210,50,10);
      checkStatus = new Label("Server werden überprüft.......bitte
warten!");
      checkStatus.setBounds(80, 250, 300, 15);
      Button bBeenden = new Button("Beenden");
      Button bCheck = new Button("Check");
      bBeenden.setBounds(260, 280, 100, 30);
      bCheck.setBounds(40, 280, 100, 30);
      bBeenden.addActionListener(this);
      bCheck.addActionListener(this);
      add(server1);
      add(server2);
      add(server3);
      add(bBeenden);
      add(bCheck);
}

public void paint(Graphics g)
{
      g.drawString("Welche WAPSCAS-Server sind aktiv?", 90, 50);
      g.drawString("IP-Adresse:", 140, 90);
      g.drawString("Status:", 260, 90);
      g.drawString("Server1:", 60, 120);
      g.drawString("Server2:", 60, 170);
g.drawString("Server3:", 60, 220);
}
```

```
public void actionPerformed(ActionEvent event)
{
      String cmd = event.getActionCommand();
      if (cmd.equals("Beenden"))
      {
            setVisible(false);
                  dispose();
                  System.exit(0);
      }

if (cmd.equals("Check"))
      {
                  remove(server1aktiv); // Falls aktiv → entfernen
                  remove(server2aktiv);
                  remove(server3aktiv);
                  add(checkStatus); // Überprüfungsmeldung




            if(CheckServer(server1url, server1user, server1passwd)) {
                  System.out.println("Server 10.1.230.230 ist
betriebsbereit!");
            add(server1aktiv);
            }
            else {
            System.out.println("Server 10.1.230.230 ist NICHT
betriebsbereit!");
            remove(server1aktiv);
            }

            if(CheckServer(server2url, server2user, server2passwd)) {
            System.out.println("Server 10.1.230.231 ist betriebsbereit!");
            add(server2aktiv);
            }
            else {
            System.out.println("Server 10.1.230.231 ist NICHT
betriebsbereit!");
            remove(server2aktiv);
            }

            if(CheckServer(server3url, server3user, server3passwd))
            System.out.println("Server 10.1.11.105 ist betriebsbereit!");
add(server3aktiv);
            else
            System.out.println("Server 10.1.11.105 ist NICHT
betriebsbereit!");
remove(server3aktiv);

remove(checkStatus); // Entfernen der Bestätigungsmeldung
    }
  }
}
```

3.3 CheckServer-Methode in Wapscas Hauptklasse

Die Methode CheckServer() versucht eine Verbindung zur Datenbank aufzubauen und liefert bei Erfolg true zurück. Sollte dabei ein Fehler auftreten, wird false zurückgeliefert, was soviel bedeutet, wie nicht funktionsfähig.

```
public static boolean CheckServer(String url, String user, String passwd)
{
      try {
                  Class.forName(driver);
                  Connection con =
DriverManager.getConnection(url,user,passwd);
                  con.close();
                  return true;
      }

      catch (ClassNotFoundException cnfe) {
                  // Exception werden nicht zusätzlich noch ausgegeben.
                  return false;
      }
      catch (SQLException se) {
                  return false;
      }
}
```

3.4 Serverüberprüfung am Handy

Dieses Servlet wird vor dem Kellnerlogin gestartet und dient dazu, um zu überprüfen, ob der ausgewählte Server funktionsfähig ist oder nicht.

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class WMLServerCheck extends HttpServlet {

      public void doGet(HttpServletRequest request, HttpServletResponse
response)
                  throws ServletException, IOException
      {
            String driver = "org.gjt.mm.mysql.Driver"; //MySQL-JDBC-Treiber
                  String url = "jdbc:mysql://127.0.0.1/WAPSCAS";
                  String user = "root";   //MySQL-Benutzername
                  String passwd = "";           //MySQL-Passwort

                  response.setContentType("text/wml");
            PrintWriter out = response.getWriter();
            StingBuffer wml = new StringBuffer();
            wml.append(Wapscas.WMLHead() +
            "<wml>\n<card id=\"Servercheck\" title=\"WAPSCAS-
Servercheck\">\n");

                  if(Wapscas.checkServer(url, user, passwd)
                  {
```

```
                        wml.append("<p align=\"center\"> Server ist
funktionsfähig!" +
"<br/><a href=\"" + Wapscas.server + Wapscas.port1 +
"/wml/index2.wml\">Weiter</a>\n" +
                        "</p>\n</card>\n</wml>");
            }

                  else
                  {
                  wml.append("<p align=\"center\"> Server ist NICHT
funktionsfähig!\n" +
                          "<br/>Wählen sie bitte einen anderen Server.\n" +
                          "</p>\n</card>\n</wml>");
                  }

                  out.println(wml.toString());
      }
}
```

Die Methode CheckServer() in der Hauptklasse Wapscas, wird als if-Bedingung verwendet. Ist der Server funktionsfähig, liefert diese Methode true zurück und am Handydisplay erscheint die Meldung „Server ist funktionsfähig“. Mit dem Link „Weiter“ gelangt der Kellner dann zum Login.

Liefert CheckServer() false zurück, ist der Server nicht funktionsfähig und die Fehlermeldung „Server ist NICHT funktionsfähig! Wählen sie bitte einen anderen Server.“ erscheint.

3.5 MySQL-Konfiguration

- Zuerst muss sichergestellt werden, dass auf dem Master und dem Slave eine MySQL-Datenbank ab Version 3.23.15 installiert ist (3.23.33 verwendet). Es wird empfohlen, auf beiden Rechnern die selbe Version zu verwenden.

- Anschließend muss am Master ein User mit dem FILE Privileg angelegt werden. Um so einen User names repl mit dem Passwort repl anzulegen, ist es notwenig in der MySQL-Console folgenden Befehl einzugeben:
  *GRANT FILE ON *.* TO repl@“%“ IDENTIFIED BY ´repl´;*

- Master herunterfahren

- Danach muss ein Snapshot vom gesamten data-Verzeichnis (c:\mysql\data) gemacht werden.

- Anschließend müssen folgende Einträge in die Datei my.ini erfolgen:
  [mysqld]
  log-bin
  server-id=1

- Starten des Masters

- Hinzufügen der folgenden Zeilen in my.ini am Slave:
  [mysqld]
  master-host=10.1.230.231
  master-user=repl

  master-password=repl

  master-connect-retry=1
  server-id=2

- Snapshot des Masters auf Slave kopieren

- Slave neu starten

Mit dieser Konfiguration ist nun ein Rechner der Master und der andere der Slave. Wird also am Slave ein Update durchgeführt, so ändert sich dadurch am Master nichts. Um nun Updates in beide Richtungen zu gewährleisten, werden beide Rechner so konfiguriert, dass jeder von ihnen Slave und Master zugleich ist.

master-connect-retry: Anzahl der Sekunden, die der Slave wartet, bevor er versucht eine erneute Verbindung aufzubauen (im Falle eines Masterabsturzes).

4 Ergebnisse

4.1 Java-Applikation

Gestartet wird das Programm in der Console:

java ServerCheck

Abbildung 5

Nach dem Start erscheint dieses Fenster. Durch Betätigung des Check-Buttons erscheint die Meldung „Server werden überprüft.....bitte warten!“ und die Überprüfung der Server wird gestartet.

Besteht eine Netzwerkverbindung zum jeweiligen Server, ist die Überprüfung nach kurzer Zeit abgeschlossen. Der Server wird als aktiv markiert, wenn die MySQL-Datenbank gestartet ist.

Besteht jedoch keine Netzwerkverbindung kann die Überprüfung bis zu 30 Sekunden pro Server dauern. Ein Timeout, um diese Zeit zu verkürzen, konnte leider nicht gesetzt werden.

Abbildung 6

4.2 Server-Redundanz

Nachdem beide Rechner so konfiguriert wurden, wie oben erwähnt, scheint in der Prozessliste folgendes auf:

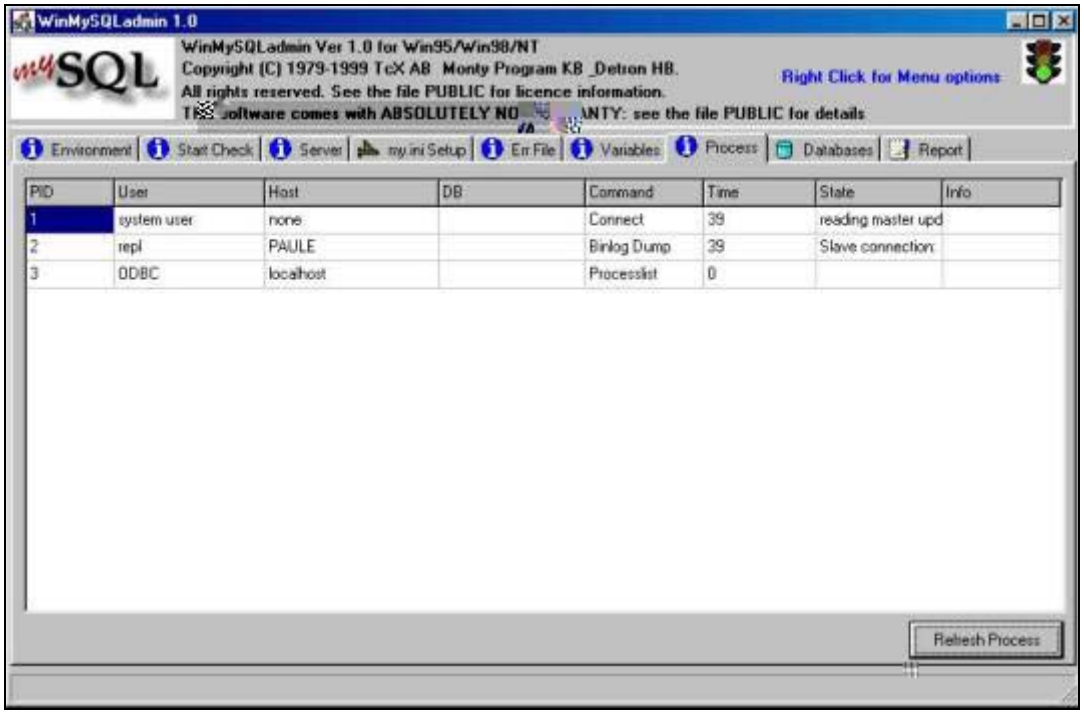

Abbildung 7

reading master updates:

Die Datenbank wartet auf neue Master-Updates. Erfolgt nun ein Update am Master, werden die selben Updates auch am Slave ausgeführt.

Slave connection:

Da der Slave auch ein Master ist, muss er natürlich auch ein Slave-Connection zur Verfügung stellen.

Im MySQL Data-Verzeichnis (c:\mysql\data) werden 3 neue Dateien angelegt:

wapscas-bin.001 (wapscas → Rechnername)

wapscas-bin.index

master.info

wapscas-bin.001 ist die erste „binary-log" Datei und enthält folgendes:

```
3.23.33-debug-log
```

wapscas-bin.index enthält die Namen, aller bereits erstellten „binary-log" Dateien. Anfangs nur:

```
.\wapscas-bin.001
```

master.info enthält Informationen zum Master:

`wapscas2-bin.001`	*letzte Binärdatei*
`10.1.230.231`	*IP-Adresse des Masters*
`repl`	*Benutzername*
`repl`	*Passwort*
`3306`	*Port*
`1`	*master-connect-retry*

Bei jedem Neustart der Datenbank wird eine neue Binärdatei erzeugt (wapscas-bin.002 – wapscas-bin.n) und in die Indexdatei wapscas-bin.index hinzugefügt.

4.2.1 Testapplikation

Diese Testapplikation dient dazu, um einige aufeinander folgende Datenbankupdates zu simulieren.

```
import java.sql.*;

public class InsertData
```

```
{
      public static void main(String[] args)
      {
            String driverClass = "org.gjt.mm.mysql.Driver";
            String url = "jdbc:mysql://127.0.0.1/WAPSCAS";

            String Kellner = "CREATE TABLE Kellner " +
                        "(kellner_nr INTEGER NOT NULL AUTO_INCREMENT, " +
                        "vorname char(15), " +
                        "nachname char(15), " +
                        "passwd char(6), " +
                        "primary key (kellner_nr));";
            String Speisekarte = "CREATE TABLE Speisekarte " +
                        "(name char(30) NOT NULL, " +
                        "art char(3), " +
                        "preis SMALLINT, " +
                        "PRIMARY KEY (name));";
            String Auftrag = "CREATE TABLE Auftrag " +
                        "(auftrag_nr INTEGER NOT NULL AUTO_INCREMENT, " +
                        "name char(30), " +
                        "anzahl INTEGER NOT NULL, " +
                        "fertig BIT DEFAULT 0, " +
                        "tisch_nr smallint," +
                        "bezahlt BIT DEFAULT 0, " +
                        "PRIMARY KEY (auftrag_nr)," +
                        "FOREIGN KEY (name) REFERENCES Speisekarte);";

try {
                  Class.forName (driverClass);
            }
catch (ClassNotFoundException exc) {
                  System.out.println(exc.getMessage());
                  System.exit(1);
            }

            try {
                  Connection con =
DriverManager.getConnection(url,"root","");
                  Statement stmt = con.createStatement();

                  stmt.executeUpdate(Kellner);
                  stmt.executeUpdate(Speisekarte);
                  stmt.executeUpdate(Auftrag);
                  stmt.executeUpdate("insert into Kellner values(null,
\"Raimund\", \"Antonitsch\", \"abc\")");
                  stmt.executeUpdate("insert into Kellner values(null,
\"Hafner\", \"Dietmar\", \"def\")");
                  stmt.executeUpdate("insert into Kellner values(null,
\"Günther\", \"Hölzl\", \"tux\")");
                  stmt.executeUpdate("insert into Kellner values(null,
\"Martina\", \"Lausegger\", \"raimi\")");
                  stmt.executeUpdate("insert into Speisekarte
values(\"Wienerschnitzel\", \"hsp\", 85)");
      stmt.executeUpdate("insert into Speisekarte values(\"Gulasch\",
\"hsp\", 75)");
```

```
                    stmt.executeUpdate("insert into Speisekarte
values(\"Indianer mit Schlag\", \"nsp\", 35)");
                    stmt.executeUpdate("insert into Speisekarte
values(\"Tomatensuppe\", \"vsp\", 40)");
                    stmt.executeUpdate("insert into Speisekarte values(\"Bier
0,5l\", \"gtr\", 28)");
                    stmt.executeUpdate("insert into Speisekarte values(\"Cola
0,3l\", \"gtr\", 25)");
                    // 40 weitere Inserts folgen noch
              }
              catch (SQLException exc) {
                    System.out.println("SQLException: " + exc.getMessage());
              }
        }
}
```

Zunächst wird eine Verbindung zur Datenbank aufgebaut. Anschließend werden die 3 Tabellen erzeugt. Danach folgen 50 aufeinanderfolgende Inserts.

4.2.2 Kommunikationsaufwand

Mit Hilfe des Fluke Protocol Inspectors wurde der Kommunikationsaufwand, der aufgrund der Testapplikation zwischen den Server entstand, ermittelt:

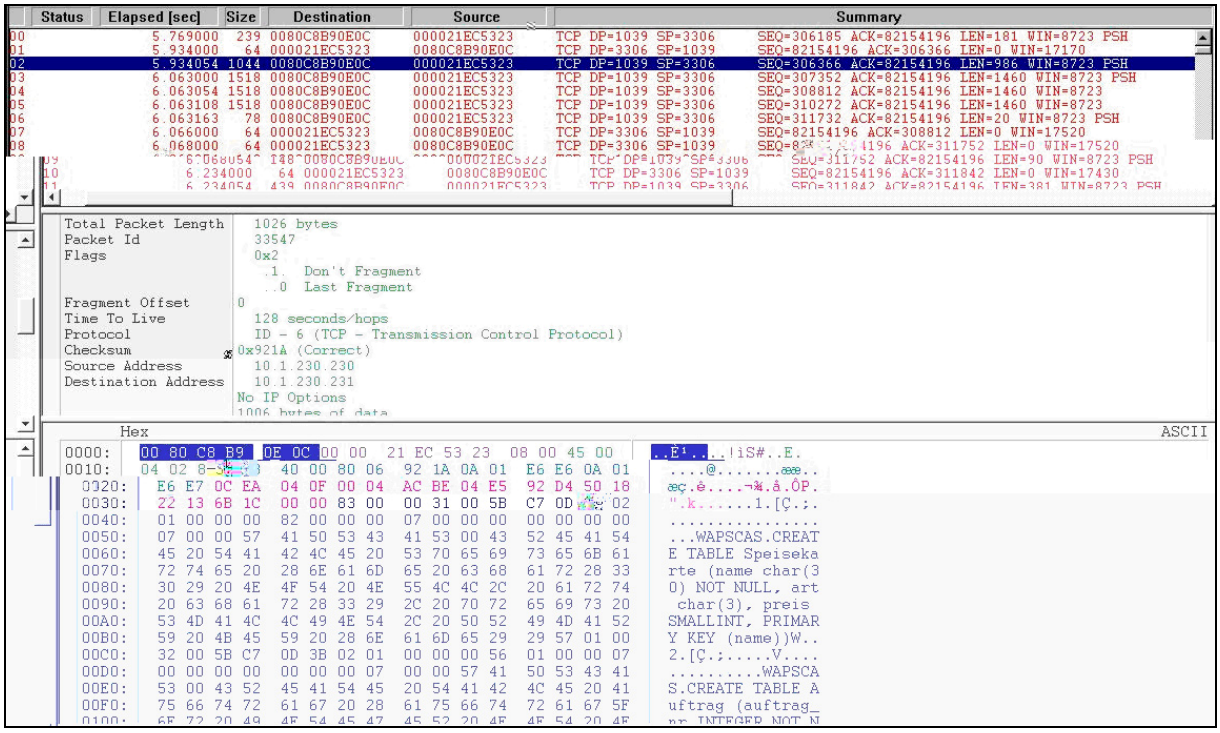

Abbildung 8

Der Master sendet das erste Packet von Port 3306 zum Slave auf Port 1039. Nach 1-5 Packet(en) sendet der Slave eine Art Acknowledge (immer 64Byte lang) von Port 1039 zum

Master auf Port 3306. Ein System, bezüglich der Anzahl der empfangenen Pakete bevor das Ack gesendet wird, ist nicht zu erkennen.

Um die 50 Updates vom Master zum Slave zu übertragen, vergeht weniger als 1 Sekunde.
Wie aus der ASCII-Ansicht ersichtlich, werden reine SQL-Statements unverschlüsselt übertragen.

Der Kommunikationsoverhead ist durch die Übertragung der SQL-Statements an sich schon gering, könnte aber theoretisch durch Substitution oft verwendeter Befehle (INSERT, DELETE,...) verringert werden.

Ca. 39% der Pakete haben ein Größe von 64 Byte und sind daher Acks vom Slave. Die anderen Pakete werden vom Master an den Slave gesendet und variieren je nach Größe des SQL-Statements.

5 Schlussfolgerung

Auf dem Gebiet der betriebssicheren Client/Serverkopplung gibt es, wie im theoretischen Teil bereits erwähnt, viele Lösungswege. Der von mir realisierte Lösungsweg ist eine einfache Erweiterung zum ursprünglichen WAPSCAS.

Die Serverredundanz mit Hilfe der integrierten und etwas modifizierten MySQL-Funktion ist meines Erachtens nach gut geeignet, um gleiche Datensätze zu gewährleisten. Um die Übertragung jedoch vor Eindringlingen zu schützen, wäre eine Datenverschlüsselung hilfreich.

Bezüglich der Serverprüfung möchte ich folgendes sagen: Sowohl am Küchen-PC, als auch am Handy erfolgt die Überprüfung nur mit Beteiligung des Benutzers. Durch WML-Script lässt sich diese Tatsache nicht ändern. UMTS und die damit verbundenen neuen Möglichkeiten, eventuell Embedded Java, würden es ermöglichen, die Serverprüfung und die Serverwahl zu automatisieren.

Eine automatische Prüfung und Wahl des Servers am Küchen-PC bereits schon realisiert werden (siehe 2.2.1).

Anhang B:

Grundlagen der Diplomarbeit

Sprachverarbeitung bei elektronischen Bestellsystemen

Bearbeiter: Dietmar Hafner

Inhaltsverzeichnis

1 Aufgabenstellung 3
1.1 Beschreibung der Grundlagen der Diplomarbeit 3
1.2 Englische Beschreibung 3
2 Theoretische Erörterungen: 4
2.1 Grundlegendes 4
2.2 Was ist die JAVA Speech API (Application Programming Interface)? 4
2.3 Ziele der JAVA Speech API 4
2.4 JAVA Applikationen mit Sprachunterstützung 5
2.5 Anwendungsgebiete für Sprachunterstützung 5
2.5.1 Desktop 5
2.5.2 Telefonsysteme 6
2.5.3 PDA (Personal Digital Assistant) 6
2.5.4 Sprachverarbeitung im Internet 7
2.6 Implementationen 7
2.7 Hardware und Software Anforderungen 8
2.8 JAVA Speech Markup Language (JSML) 9
2.8.1 JSML Dokument Struktur 11
2.8.2 JSML Elemente 11
2.8.2.1 SAYAS Element 12
2.8.2.2 PROS Element 13
2.9 JAVA Speech API Grammar Format (JSGF) 13
3 Praktische Umsetzung 14
3.1 Lösungswege 14
3.2 Entwicklungstools 15
3.2.1 JAVA Developer Kit (JDK) 15
3.2.2 RealJ 16
3.2.3 JAVA Speech APIs (JSAPIs) 16
3.2.3.1 IBM's Speech for JAVA (Details siehe [2]) 16
3.2.3.2 Jesse Peterson JAVA Speech API Implementation (Details siehe [3]) 17
3.2.3.3 Cloudgarden's Java Speech API 1.0 Implementation for Windows Platforms (Details siehe [4]) 17
3.2.3.4 Microsoft's Speech SDK 5.0 (Details siehe[5]) 18
3.2.3.5 Implementation von Erich Hambuch (Details siehe [6]) 18
4 Ergebnisse 18
4.1 JAVA Speech Programmierung 18
4.2 Graphische Programmierung 22
4.2.1 Programmteil: 22
4.3 Datenbankzugriff: 24
4.3.1 Programmteil: 24
4.4 Ressourcenbedarf 26
5 Schlussfolgerungen 26
6 Literaturverzeichnis 28

1 Aufgabenstellung

Als Erweiterung zum bestehenden Projekt in dem Unterrichtsfach Labor sollte eine Client/ Serveranwendung zur akustischen Ein/Ausgabe am Webclient realisiert werden. Es ist nun zu analysieren, welche Sprachmodelle verwendetet werden können und wie hoch der Ressourcen Bedarf ist. Wie können die Systeme parametrisiert werden und wie hoch liegt der Aufwand bei der Erstinstallation?

1.1 Beschreibung der Grundlagen der Diplomarbeit

Grundsätzlich geht es darum, zusätzlich zur Auftragsanzeige (im Webinterface) eine akustische Ergänzung zu erschaffen. Diese sollte die aktuellen Aufträge, die vom Kellner per WAP Handy bestellt wurden, in Sprache umsetzen und mit Hilfe von Lautsprechern wiedergeben.

Da nun das Küchenpersonal bereits die Möglichkeit hat, erfüllte Aufträge zu löschen, tritt dabei eine Spracherkennung in Kraft, die Sprachbefehle verarbeiten kann und die erfüllten Aufträge löscht.

Folgender Nutzen sollte dadurch erzielt werden:

- Flexiblere Arbeitstätigkeit für das Küchenpersonal.
- Wegersparnis (PC – Küchengerät), da die Audiogeräte nicht direkt am PC angebracht sein müssen.

Weiters gilt es festzuhalten, wie hoch der Ressourcenbedarf bei solch einer Applikation ist (im Besonderen die Speicherbelegung). Auch die Verwendung von Sprachmodellen ist in der Anwendung enthalten.

Um eine Sprachverarbeitung einzusetzen, wird die Entwicklungsumgebung zur Erstellung dieser Anwendung auf JAVA festgelegt, da JAVA Sun eine Sprach-Technologie entwickelt hat (siehe Punkt 2).

Auch die Oberfläche der Anwendung wird in weiterer Folge mit JAVA programmiert.

1.2 Englische Beschreibung

My project is about electronic voice processing used for our „Jahreslaborprojekt“ called WAPSCAS (wirelesss application supported catering accounting system). It should symplify the events which can occur in a kitchen, where our Webinterface is used to show the current orderings to the the kitchen’s staff. An application which “speaks” the current ordering should be implemented. A speech recognition is used to interprete commands from the staff e.g. after finishing ordering to delete them.

2 Theoretische Erörterungen:

2.1 Grundlegendes

Sprachverarbeitung wurde in der letzten Zeit ein immer interessanter werdendes Merkmal, welches PCs von heute besitzen. Da die Leistung der heutigen PCs für eine Verarbeitung ständig wuchs wurde die Sprachsynthese dadurch entscheidend verbessert. Real-time Anwendungen in Zusammenhang mit Sprache sind somit möglich geworden. In besonderen die Anpassung der Sprachausgabe (Verwendung von genau deklarierten Sprachmodellen) war ein entscheidener Schritt in Richtung natürlich klingender Sprachen.

Andererseits wurde die Spracherkennung auch interessanter, da mit steigender PC Leistung, die Fehlerhäufigkeit sank und „einfaches“ Diktieren nun möglich macht. Spracherkennungsprogramme, wie IBMViaVoice werden heute bereits auf Heim PCs eingesetzt und erleichtern somit die Arbeit. Aber auch Sprachbefehlsverarbeitung ist ein interessanter Aspekt, der es möglich machen wird, dem Computer kurze Befehle und 7verständliche Anweisungen zu erteilen.

2.2 Was ist die JAVA Speech API (Application Programming Interface)?

Die JAVA Speech API ist ein Softwareinterface, welches ermöglicht, Sprachverarbeitungstechnologie auf PCs einzusetzen. Es gehört zur Familie der JAVA Media API (JAVA 2D API, JAVA 3D API, JAVA Sound API sowie noch weitere)

Es gibt zwei Kernbereiche die unterstützt werden:

- Spracherkennung (Computer verstehen die menschliche Sprache)
- Sprachwiedergabe (TTS Systeme (Text to speech))

JAVA Speech API:

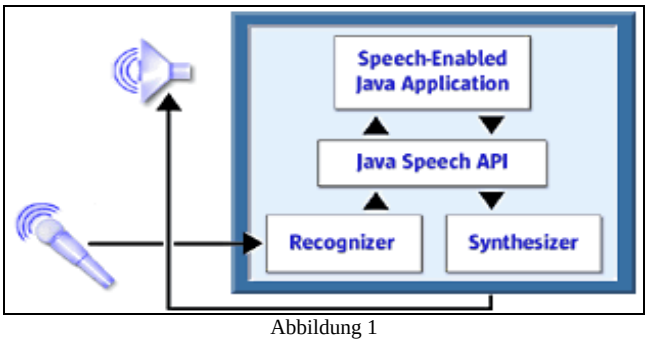

Abbildung 1

2.3 Ziele der JAVA Speech API

Da es möglich ist, das Sprachinterface mit anderen JAVA Applikationen in Verbindung zu bringen, ergeben sich folgende Ziele:

- Plattformübergreifende Sprachwiedergabe und Erkennung per JNI (JAVA Native Interface).
- Integration von anderen JAVA Media APIs
- Einfachheit
- Verbesserung der Natürlichkeit durch eine eigens geschaffene Sprache (JSML, siehe später)

2.4 JAVA Applikationen mit Sprachunterstützung

Da JAVA folgende Eigenschaften enthält, werden diese auch an sprachunterstützte Applikationen weitergegeben:

- Portabilität der APIs, JAVA Virtual Machines sind für nahezu jede Hardware und Betriebssystemkonfiguration erhältlich.
- Objektorientierte Programmierung.
- Sicherheit in Zusammenhang mit Netzwerken.
- Synchronisation mit anderen JAVA Objekten (JAVA AWT)

2.5 Anwendungsgebiete für Sprachunterstützung

Sprachtechnologie ermöglicht eine weitere Interaktion zwischen Mensch und PC. Da oftmals die Hände für andere Aufgaben frei sein müssen, kommen Applikationen mit Sprachunterstützung gerade recht. Sprachwiedergabe und Erkennung kann auch die Arbeitsmöglichkeiten für behinderte Menschen erweitern, sowie die negativen Eigenschaften von PC-Arbeiten (Augenbeschwerden und Haltungsschäden) reduzieren. Da die Entwicklung von sprachunterstützten Anwendungen rasant vor sich geht, kann man in nächster Zeit mit einer Verbesserung hinsichtlich der Computer-Mensch Interaktion rechnen. Ein paar Beispiele für den Einsatz von Sprachtechnologie werden beschrieben:

2.5.1 Desktop

Sprachverarbeitung kann die graphische Oberfläche eines Programms oder eines Betriebssystems verbessern. Im einfachsten Falle werden Abfragen wie JA/ NEIN/ ABBRECHEN durch Sprachbefehle ersetzt. Bei aufwendigen Beispielen (z.B. Schriftart-, Schrifttyp und Schriftgrößeänderung) können dadurch einige Mausklicks und Menüselektionen gespart werden.

Bei CAD (Computer Aided Design) kann die Arbeitsperformance wesentlich gesteigert werden, da mehrere Sachen gleichzeitig ausgeführt werden können (Objekte markieren und ziehen und per Sprache gleichzeitig die Farbe ändern).

Ein weiterer Vorteil liegt in der Erlernbarkeit von Programmen. Für viele Anwender ist es schwierig sich Menüpunkte zu merken. Sprachbefehle anzuwenden ist um einiges flexibler, da sie einfacher zu merken sind.

Mittlerweilen gibt es Diktier- oder Spracherkennungsprogramme die mehr als 100 Wörter pro Minute erkennen und eine Erkennungsrate von bis zu 95% aufweisen. Diese Werte übersteigen die „Tipp-Fähigkeiten“ der meisten Anwender.

Sprachwiedergabe kann Vorteile bei der Korrektur von Texten bieten. Fehler werden einfacher erkannt, wenn ein Text vorgelesen wird, anstatt ihn am PC durchzulesen. Auch Hintergrund Aktionen (wie Statusberichte von Peripheriegeräten) können wiedergegeben werden und bringen den Anwender nicht aus der Konzentration.

Es gibt bereits Anwendungen mit Sprachwiedergabe. Dabei wird die Sprachausgabe aber nicht dynamisch erzeugt, d. h. es sind die Ausgaben aufgezeichnet und belegen dadurch sehr viel Speicherplatz. Der Unterschiede zwischen dynamisch erzeugter Ausgabe und voraufgezeichneten liegt bei 1:1000 (bezogen auf den Speicherplatz).

Ein weiteres Anwendungsbeispiel für Sprachbefehlsverarbeitung wäre ein Operationssaal, wo der Chirurg keine Tastatur bedienen kann, da es ein hygienisches Risiko darstellt. Die Befehle zu sprechen hingegen stellt kein Problem dar.

2.5.2 Telefonsysteme

Remote- Abfragen stellen auch einen wichtigen Teil bei der Telephonie dar. Es gibt bereits Systeme, wo Daten per Telefonhörer abgefragt werden können. Hier findet die JAVA Speech API ebenfalls ein Anwendungsgebiet. Da die bereits vorhandenen Systemen menübasierend sind, verliert man leicht die Übersicht und kennt sich nicht mehr aus. Bei einen Spracherkennungssystem kann jedoch dies entfallen, da direkte Befehle oder Anfragen gestellt werden können. Als Beispiel ein E-Mail Remote System. Der Benutzer fragt am Telefon:“ Hab ich neue E – Mails?“ Antwort:“ Sie haben 3 neue Nachrichten ...“

2.5.3 PDA (Personal Digital Assistant)

Sprachtechnologie wird bereits in kleine handliche Geräte miteinbaut, um deren Größe weiter zu reduzieren und auf Eingabetasten verzichten zu können (z.B. Spielzeuge, Telephone handsets usw.)

Sprachtechnologie ist nahezu zwingend für solche Geräte und kann auch die immer steigende Performance und Rechenleistung nutzen. Wie bereits erwähnt können Tastatur oder Eingabetasten (Spracherkennung) aber auch große Displays (Sprachwiedergabe) weggelassen werden. Ein Anwendungsgebiet für JAVA wäre dann das EmbeddedJAVA oder PersonalJAVA, welches auf Handheld Computern oder Geräten dergleichen eingesetzt wird.

2.5.4 Sprachverarbeitung im Internet

Die JAVA Speech API ermöglicht Applets über das Internet oder Intranet zu übertragen, welche dem Anwender PC Sprachfähigkeiten verleiht. Zum Beispiel können Internet Seiten nun beim „Surfen“ vorgelesen werden, anstatt sie zu lesen. Spracherkennung kann wiederum den Internetbrowser Sprachbefehle vermitteln und verbessert dadurch das WWW. Sprachwiedergabe kann auch hier als Statuswiedergabe eingesetzt werden. Z.B.: Status von Downloads oder vom Status des Seitenaufbaus.

Die JAVA Speech API Sicherheitseigenschaften legen auch fest, dass Applets dem Client nicht seine Ressourcen „wegfressen“ dürfen oder gar den PC zum Absturz bringen. Es können auch Abfragen angebracht werden, bevor z.B. ein Spracherkennungsapplets initialisiert wird, ob der Benutzer dies wünscht und genügend Ressourcen zur Verfügung hat.

2.6 Implementationen

Die JAVA Speech API wurde von mehreren Firmen entwickelt. Folgende Firmen waren daran beteiligt:

- Apple Computers
- AT&T
- Dragon Systems
- IBM Coperation
- Novell Inc.
- Philips Speech Processing
- Texas Instruments Inc.

Für die Festlegung des Standards, „JAVA Speech“ ist SUN Microsystems verantwortlich.

Es gibt mehrere Mechanismen für die Implementierung der JAVA Speech APIs:

Ursprüngliche Implementationen:

Die meisten Implementationen sind in C oder C++ realisiert. Das heisst, sie sind plattformabhängig. Als Bespiele dafür können Apple Speech Managers und Microsoft’s Speech API (SAPI) angesehen werden.

Die JAVA Speech API kann meist auch als Ergänzung zu bestehender Software implementiert werden (Verwendung von JNI und JAVA Software Wrappers)

JAVA Software Implementationen:

Sprachsynthesizer können auch mit JAVA Software geschrieben werden. Diese profitieren von JAVAs Portabilität, Plattformunabhängigkeit sowie auch von der Verbesserung der Ausführungsgeschwindigkeit von JAVA Virtual Machines.

Telephonie Implementationen:

Enterprise Telephonie Anwendungen werden bereits in reiner Hardware realisiert. Es müssen sehr viele Verbindungen unterstützt werden und dadurch gibt es oft Ausführungen mit sog. DSP Karten. Sprachtechnologie kann auch hier eingesetzt werden in Form der JAVA Speech API in Verbindung mit JAVA Software.

2.7 Hardware und Software Anforderungen

Um die JAVA Speech API verwenden zu können, muss ein Minimum an Software und Hardware verfügbar sein.

Sprachverarbeitungssoftware:

Ein JSAPI arbeitsfähiger Sprach- Recognizer und Synthesizer wird benötigt.

Systemanforderungen:

Es wird ein leistungsfähiger PC vorrausgesetzt, um die Sprachwiedergabe und Erkennung nutzen zu können. Genügend Prozessorleistung, Speicher und Festplattenspeicher sollte schon vorhanden sein.

Audio Hardware:

Sprachwiedergabe benötigt einen Audio Output. Spracherkennung einen Input. Die meisten PCs, die heutzutage am Markt sind, sind mit Soundunterstützung ausgestattet. Für Diktiersysteme wird jedoch eine qualitativhochwertigere Soundkarte vorrausgesetzt.

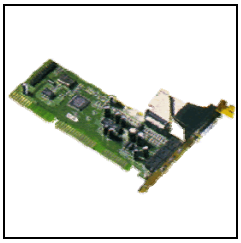

Abbildung 2

Mikrophone:

Spracherkennungssysteme benötigen ein Mikrophon als Audio Input. Manche Sprach-Recognizer, vorallem Diktiersysteme, reagieren sensibel auf die Art des Mikrophons. Headset bieten am meisten Performance, auch wenn sie bei lauten Umgebungsgeräuschen eingesetzt werden. Tisch Mikrophone sind für restliche Anwendungen akzeptabel.

Abbildung 3

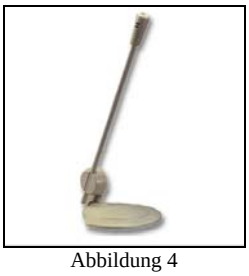

Abbildung 4

2.8 JAVA Speech Markup Language (JSML)

JSML ist eine textspezifische Ausführung, die den Input für einen Sprach Synthesizer darstellt. Sie wurde entwickelt von den Firmen, die bereits unter Punkt 2.6 angeführt wurden. Die Standardisierung ist noch nicht abgeschlossen. Es kann somit noch mit Veränderungen hinsichtlich der Dokument Struktur gerechnet werden.

Ein Sprach-Synthesizer verleiht einen Computer die Fähigkeit zu sprechen. Eine Applikation stellt diesem Text zur Verfügung, welcher von ihm in ein Audio Signal umgewandelt wird.

Sprachwiedergabe:

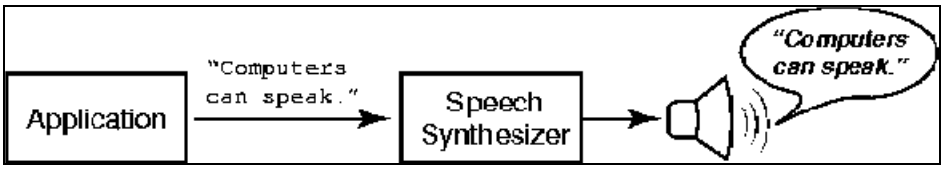

Abbildung 5

Sprach-Synthesizer werden entwickelt, um die Natürlichkeit der Sprache nachzubilden. Im Gegensatz zum Menschen versteht ein Sprach Synthesizer nicht, was er sagt.

JAVA Speech Markup Language verleiht einen Text, welcher gesprochen werden sollte, zusätzliche Eigenschaften, um die Qualität und Natürlichkeit der künstlichen Sprache zu verbessern. Es wird auch erlaubt, die Kontrolle über die „Produktion“ von einer künstlichen Sprache zu übernehmen (Betonung und Aussprache von Wörtern, Platzierung von Pausen, Tonhöhe, usw.)

JSML wurde soweit entwickelt, dass so gut wie alle Sprachen unterstützt werden (Japanisch, Englisch, Deutsch, Spanisch, Französisch usw.). Die Anwendungsgebiete sind reichhaltig, von E-Mail Systemen bis hin zu Web Pages.

Beispiel für JSML:

<SENT> Computers <EMP> können </EMP> sprechen </SENT>

Es ist eine Ähnlichkeit mit HTML Tags ist erkennbar, da beide Formate, dieselben „Vorfahren“ haben: SGML (Standard Generalized Markup Language). HTML ist spezialisiert für eine visuelle Darstellung, JSML für Sprachinformationen.

Sprachwiedergabe mit JSML:

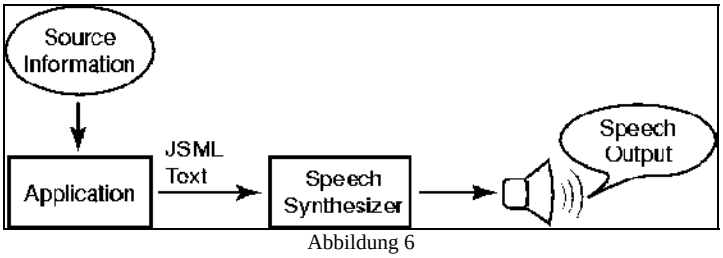

Abbildung 6

Die Applikation ist dafür verantwortlich, dass die Quellinformation entsprechend in JSML Text umgewandelt wird. Ein Webbrowser zum Beispiel könnte die Fähigkeit erlangen Web Seiten vorzulesen, wenn sie dementsprechend in JSML Text konvertiert werden kann. Diese würde eine Umwandlung von HTML (Hyper Text Markup Language) in JSML darstellen.

JSML Elemente sind entweder sog. Container Objekte (8 Stück) oder leere Elemente (siehe später). Ein Container Element besteht aus einen Start Tag (<SENT>) und einen End Tag (</SENT>). Der Text zwischen Start und End Tag stellt den Container Text dar. Der Start Tag definiert das Element und kann auch eines oder mehrere Attribute enthalten. Siehe folgendes Bild:

Elemente und Attribute

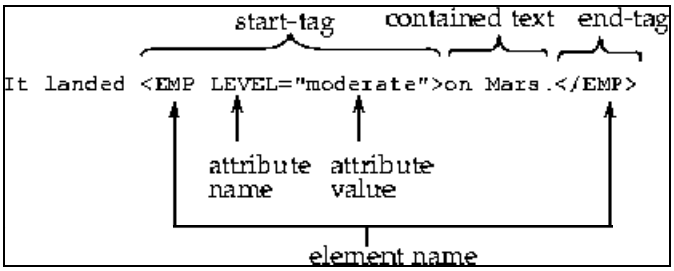

Abbildung 7

Es gibt auch wie erwähnt, leere Elemente. Diese enthalten nur einen Start Tag und keinen weiteren Text.

Beispiel:

Ein lautes Geräusch, <BREAK SIZE="large"/>und dann wurde es still.

Bei undefinierten Namen kann es jedoch zu Problemen kommen, da der Synthesizer nicht weiß, wie es diesen wiedergeben sollte. (z.B.: URLs oder sonstiges)

2.8.1 JSML Dokument Struktur

JSML basiert auf XML (Extensible Markup Language), welcher ein einfacher „Dialekt“ von SGML ist. Ein JSML Dokument sieht folgendermaßen aus:

```
<?XML version="1.0" encoding="UCS-2"?>
<JSML>
```

und endet mit:

```
</JSML>
```

Es können auch Escape Sequenzen innerhalb eines Dokuments eingesetzt werden.

Um den JSML Text richtig zu übersetzen kann ein einfacher XML Parser verwendet werden, um dem Synthesizer den Inhalt zu übermitteln. Kommentare sind ebenfalls zulässig.

2.8.2 JSML Elemente

Wie bereits erwähnt ist der Inhalt eines JSML Dokuments zusammengesetzt aus unterschiedlichen Elementen (Struktur, Produktion und Verschiedenartige). In der folgenden Tabelle werden diese aufgelistet und genauer beschrieben.

Element Funktion	Element Name	Element Typ	Element Beschreibung
Struktur	PARA	Container	Legt fest, dass der Text innerhalb des Containers ein Paragraph ist.
	SENT	Container	Legt fest, dass der Text innerhalb des Containers ein Satz ist.
Produktion	SAYAS	Container	Legt fest, wie der Text zu sprechen ist.
	EMP	Beides	Legt fest, wie der Text betont werden muss.
	BREAK	Empty	Legt eine Pause fest.
	PROS	Container	Legt satzrhythmische Eigenschaften fest, wie die grundlegende Tonhöhe, Volumen für den Text innerhalb des Containers.
Miscellaneous	MARKER	Empty	Löst etwas aus, wenn die Sprache diesen Punkt erreicht.
	ENGINE	Container	Instruktionen für einen bestimmten Sprach Synthesizer.

Es werden nun nur die wichtigsten bzw. interessant erscheinenden Elemente beschrieben.

2.8.2.1 SAYAS Element

Ist ein wichtiges Element, da damit festgelegt werden kann, wie ein Text zu sprechen ist. Es gibt wieder Attribute zu diesem Element wie: SUB, CLASS, PHON und MARK.

SUB : Ist für Ersatz von Abkürzungen zu gebrauchen.

CLASS: Hat mehrere Werte, wie DATE, DIGIT, LITERAL, NUMBER und TIME. Wichtig damit der Synthesizer spezielle Formate richtig interpretiert und ausspricht.

PHON: Da es eine internationale Lautschrift gibt, kann diese mittels diesem Attribut eingesetzt werden.

2.8.2.2 PROS Element

Ein ebenfalls wichtiges Element, dass das Sprachsignal beschreibt. Dazu gehören:

RATE: Gibt die Anzahl der Wörter pro Minute an (Sprachgeschwindigkeit, Werte n)

VOL: Volumen (0.0 bis 1.0)

PITCH: Damit kann die Grundtonhöhe gesetzt werden.

RANGE: Legt den Bereich der Tonhöhe fest. (Angabe in Hertz)

MARK: Wenn dieses Element erreicht wird, kann damit etwas ausgelöst werden.

Das folgende Bild stellt den Wert für die Tonhöhe (Pitchrate) dar. Die Pitch baseline (Grundtonhöhe) liegt bei Frauen um die 140 bis 280Hz (Range 80Hz+) und bei Männern bei 70 bis 140Hz (Range 40-80Hz).

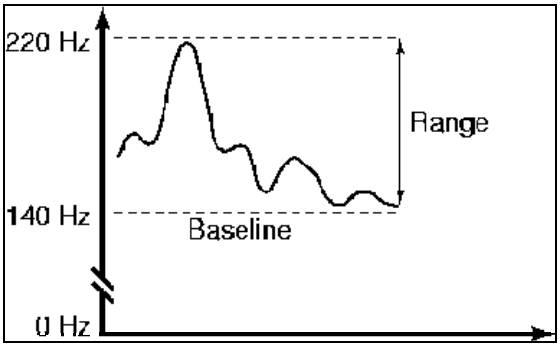

Abbildung 8

2.9 JAVA Speech API Grammar Format (JSGF)

Spracherkennungssysteme verleihen Computern die Fähigkeit, der Sprache des Anwenders zu lauschen und festzustellen was gesagt wurde. Bisher wurden aber noch keine Systeme entwickelt, die auch den Inhalt des erkannten Text verstehen. Um nun eine Erkennungsgenauigkeit zu erreichen, verwenden Spracherkennungssysteme eine eigene Grammatik.

JSGF hat eine Möglichkeit geschaffen, diese Grammatik mit entsprechenden Regeln einzusetzen.

Eine Sprachregel (Rule grammar) definiert die „Äußerungen“ eines gesprochen Textes. Eine „Äußerung“ kann auch als Befehl angesehen werden. z.B.: Datei öffnen, Fenster schließen.

Für weitere Details zu dieser Spracherkennungssprache siehe Punkt 5.1 [1]

3 Praktische Umsetzung

3.1 Lösungswege

Einerseits, wurde die Sprachverarbeitungssoftware oder besser gesagt die Entwicklungsumgebung schon am Anfang festgelegt. Es galt nun eine Applikation zu entwickeln, die die aktuellen Aufträge (gespeichert in der Datenbank und im Webinterface angezeigt) zu einer Sprachausgabe verarbeitet.

1. Eine eigenständige Applikation wird entworfen, die mittels Konsolenbefehl oder Batch Datei gestartet wird. Die Applikation sollte graphische Elemente enthalten, um den Anwender ein gewisses Feedback zu liefern.
   Da nun einerseits das Webinterface im Internetbrowser angezeigt wird, müsste ein weiteres Programm (die Sprachwiedergabe Applikation) zusätzlich laufen. Diese Anwendung muss auf die Daten des Datenbank Servers zugreifen können, da man sich auf der „Clientseite“ befindet.
   Mögliche Probleme: Die Programme am Bildschirm verwirren den Anwender, wenn eventuell einmal im Webinterface nachgeschaut wird, welche Aufträge es gibt, sowie welche noch nicht bezahlt wurden. Weiters kann es zu Schwierigkeiten kommen, wenn der Internetbrowser oder die Applikation abstürzt. (beide nutzen die interne JAVA Virtual Machine). Dies hat zur Folge, wenn beide Programme abstürzen könnten muss der Anwender nun beide wieder starten.

2. Eine Integration der Applikation in das Webinterface ist auch möglich. Dies geschieht, wenn die Software als JAVA Applet programmiert wird und im Internetbrowser ausgeführt wird. Das Applet wird mittles Servlet in der Internetseite eingebunden und bekommt die Daten der Aufträge (Anzahl, Name und Tischnummer) als Parameter übergeben. Das Applet läuft ebenfalls clientseitig und muss auf die Ressourcen des Anwenders zugreifen können (wichtig). Es muss auch beachtet werden, dass immer die aktuellen Datensätze dem Applet übergeben werden.

3. Die Spracherkennungsapplikation kann wie beschrieben in Punkt 1 und 2 der Lösungswege erstellt werden. Entweder sie ist eine eigenständige Applikation oder als JAVA Applet realisiert. Ein Unterschied kann aber gemacht werden, ob die Sprachwiedergabe und Spracherkennung in ein und derselben Applikation implementiert wird.

4. Zwei selbstständige Applikationen zu entwerfen, würde nicht viel Sinn machen, das es zu Problemen mit der vom Betriebssystem bereitgestellten Ressourcen kommen kann. Wenn z.B. gleichzeitig Sprache wiedergegeben wird und erkannt wird, könnte es bei einer nicht full duplex fähigen Soundkarte zur einen Konflikt kommen.

5. Eine Applikation zu entwerfen, die zwei Funktionen besitzt, Sprachwiedergabe und Spracherkennung, würde mehr Sinn machen. Der Applikation müsste jedoch aus zwei Programmteilen bestehen und Art Synchronisation müsste Wiedergabe und Erkennung regeln.

Der Lösungsweg 2 in Kombination mit Lösungsweg 5 schien anfangs als sehr sinnvoll. Als dann jedoch Probleme auftraten, das Sprachapplet verwenden zu können, musste das Design geändert werden und eine eigenständige graphische Applikation entwickelt werden. Somit wurde Lösungsweg 1 angestrebt.

3.2 Entwicklungstools

Die Entwicklungstools sind ziemlich ähnlich denen, die beim Jahresprojekt eingesetzt wurden. Die Programmiersprache ist wiederum JAVA, da die Sprachverarbeitung ein JAVA Abkömmling ist (JAVA Speech).

3.2.1 JAVA Developer Kit (JDK)

Diese Entwicklungsumgebung für die Programmiersprache JAVA kann auf der Sun Homepage gratis heruntergeladen werden. Mittlerweilen gibt es drei weitverbreitete Versionen (1.1.8, 1.2.x und 1.3). Die Installation erfolgt per Setup. Ist lediglich noch eine Zeile in der Autoexec.bat Datei einzufügen (Win9x) oder die Umgebungsvariable unter (Windows 2000) zu setzen.

Set PATH=c:\jdk1.x\bin; (wenn es auf Laufwerk C installiert wurde).

Für die Verwendung der JAVA Speech API muss noch eine weitere Veränderung hinzugefügt werden.

SET CLASSPATH=c:\Dateiname.jar; (JAR Files sind ähnlich einem Zip File und enthalten oft viele kompilierte Klassendateien mit einer Verzeichnisstruktur).

Kompiliert werden JAVA Source Files in der MS-DOS Kommandozeile mit dem Befehl **javac Sourcefile.java** .

Ausgeführt werden kompilierte Class Dateien mit dem Befehl **java Klassendateiname**.

Zu bemerken ist noch, dass die Version von der JDK, *1.2.2* ist. Diese enthält bereits die JAVA Runtime Environment. Diese Version wurde eingesetzt, da es bei anderen Versionen zu Problemen mit der JAVA Speech API kam.

3.2.2 RealJ

Die JAVA Sourcefiles können mit diesem Programm erstellt und editiert werden. Eine bessere Übersicht und Verwaltung der einzelnen Sourcedateien kann durch Einsatz eines sog. Projektes im RealJ erzielt werden. Es wird auch die Möglichkeit geboten, die JAVA Dateien ohne die oben beschriebenen Kommandobefehle zu kompilieren.

Der größte Vorteil gegenüber einem anderen einfacheren Editor ist jedoch, dass die Zeilen und Spalten im Source Code angezeigt werden. Fehler zu korrigieren ist dadurch bei weitem einfacher.

3.2.3 JAVA Speech APIs (JSAPIs)

Der Standard JAVA Speech wurde von Sun Mircosystems und vielen anderen Firmen entwickelt und verbessert. Die Spezifikation dazu wurde von Sun herausgegeben. Für die Implementation einer API fühlte sich Sun nicht verantwortlich.

Zitat auf der Sun Homepage: " The Java™ Speech API (JSAPI) is not part of the JDK and Sun does not ship an implementation of JSAPI. Instead, we work with third party speech companies to encourage the availability of multiple implementations."

Angeboten werden verschiedene Implementationen (siehe Punkt 2.6) der JSAPI. Es wird nun auf diese ein wenig genauer eingegangen.

3.2.3.1 IBM's Speech for JAVA (Details siehe [2])

IBM hat eine Implementation der JSAPI veröffentlicht. Das angebotene Produkt basiert auf einen anderen IBM Produkt (IBM's ViaVoice). Es werden europäische Sprachen unterstützt wie Deutsch, Englisch, Französisch usw. Vorrausgesetzt wird eine JDK 1.1.7 mit Windows95 (32 MB RAM)

Installation und Paketumfang:

Es werden HTML Dateien mitgeliefert, die die Installation sowie die Verwendung beschreiben. Weiters gibt ein Beispiel mit dem getestet werden sollte, ob die API auch funktioniert.

Zu aller erst sollte der CLASSPATH und der PATH laut Anweisung gesetzt werden. Danach sollte des Beispielprogramm ohne Probleme funktionieren (Sprachwiedergabe und Spracherkennungsbeispiel). Abweichungen von der Sun Spezifikation sind ebenfalls enthalten.

Da IBM eine große Firma ist, wird ebenfalls ein Diskussionsforum über IBM's Speech for JAVA angeboten. Das Paket ist für den Gebrauch von 90 Tagen „released".

3.2.3.2 Jesse Peterson JAVA Speech API Implementation (Details siehe [3])

Diese API ist programmiert für die Microsoft Speech API (SAPI). Der Code verwendet die ursprüngliche Methode JNI (JAVA Native Interface) um einen Microsoft SAPI Engine anzusprechen. Momentan unterstützt SAPI nur Windows Plattformen (95, 98 und NT).

Installation und Paketumfang:

Es werden ebenfalls HTML mitgeliefert, die eine Hilfestellung bei der Installation und Verwendung darstellen sollten. Es werden für zwei JDK Versionen die Einrichtungsvorgänge beschrieben. (Classpath, Kopieren einer DLL und einer properties Datei)

Drei Beispiele werden mitgeliefert, die zum Testen dienen sollten. (TTS, HelloWorld und Speak Beispiel).

Vorraussetzungen für diese API sind: JDK ab 1.18 und Microsoft Speech API ab der Version 4.0.

Es werden ebenfalls die Abweichungen und Veränderungen von Sun's JAVA Speech API angeführt.

Wichtig ist noch, dass kein JSML unterstützt wird und keine Spracherkennung implementiert ist.

3.2.3.3 Cloudgarden's Java Speech API 1.0 Implementation for Windows Platforms (Details siehe [4])

Hier wird eine nahezu vollständige Implementation von Sun's Standard angeboten. Es wird Spracherkennung mit Grammatikregeln (JSGF) und JSML unterstützt. Die Implementation ist ein „Mischmasch" aus Microsoft SAPI 5.0 (in C++ geschrieben) und JAVA Funktionen. Von der SAPI werden einige Synthesizer Engines zur Verfügung gestellt. Die JAVA Speech Implementierung wird also durch Hilfe von Microsofts SAPI realisiert.

Installation und Paketumfang:

JDK 1.2 oder 1.3 sowie Microsoft Speech SDK 5.0 wird vorrausgesetzt. An PC Hardware wird ein PC mit Pentium II 400MHz und 256 MB als zufriedenstellende Ergebnisse liefern. Weniger Leistung mindert die Leistung bei der Spracherkennung.

Im Paket ist eine JAVA Installationsanwendung enthalten. Diese extrahiert alle gepackten Dateien sowie installiert DLL Dateien in die entsprechenden Verzeichnisse. Der CLASSPATH muss jedoch noch zusätzlich gesetzt werden.

Beispiele werden ebenfalls mitgeliefert, für Sprachwiedergabe als auch Erkennung.

Diese API ist ein kommerzielles Produkt. Es gibt eine „free“ Variante, die eine Einschränkungen besitzt: Ein „R“ wird als „W“ ausgesprochen. Dies erfolgt auch bei der Spracherkennung, wo ein R als W verstanden wird. Eine registrierte API kostet 50$. Wichtig ist noch, dass die API nur die englische Sprache unterstützt.

3.2.3.4 Microsoft’s Speech SDK 5.0 (Details siehe[5])

Ist eine gratisdownloadbare Entwicklungsumgebung für Windowsplattformen. Es werden Bespiele, ein Grammar Producer sowie andere Tools mitgeliefert. Der Einsatz dieser SDK ist jedoch daraufhin beschränkt, dass andere APIs die von ihr zur Verfügung gestellten Engines nutzen.

3.2.3.5 Implementation von Erich Hambuch (Details siehe [6])

Diese Implementation wurde ebenfalls berücksichtig sowie ausgetestet. Der Student hat eine Sprachausgabe entwickelt, die Phoneme der deutschen Sprache aneinander reiht und diese dann ausgibt. JSML und weitere Eigenschaften für den Sprachsynthesizer wurden nicht berücksichtigt.

Bei der Installation ist nur der CLASSPATH zum API Package zu setzen. Ein Testbeispiel kann auch downgeloadet werden.

Man kann nun sehen, dass mehrere APIs zur Auswahl standen. Probiert wurden alle, doch funktionierten nicht alle (siehe Punkt 5). Die ersten Versuche wurden mit der API Implementation des deutschen Studenten getätigt. Da die Sprachqualität aber sehr schlecht war, wurde folglich die Cloudgarden API eingesetzt, da sie erstens mit der SAPI von Microsoft funktioniert und JSML sowie JSGF unterstützt wird. Nachteil dieser API ist, dass es sich um eine Trial Version handelt und nur die englische Sprache unterstützt wird.

4 Ergebnisse

Im Punkt zuvor wurden nun alle Softwarepakete vorgestellt, die benötigt werden, um eine sprachfähige JAVA Applikation erstellen zu können. Informationen zur Programmierung wurden einerseits aus den mitgelieferten Beispielen als auch aus dem im Literaturverzeichnis angeführten Referenzen bezogen.

4.1 JAVA Speech Programmierung

Begonnen wurde bei der Softwareerstellunge zunächst mit dem „Sprachteil“. Es wurden so vorgegangen, wie es im JAVA Speech API Programmer’s Guide beschrieben wird.

Der erste Prototyp der meinen Namen sagen konnte sah folgendermaßen aus:

HelloWorld.java:

```
import javax.speech.*;
import javax.speech.synthesis.*;
import java.util.Locale;

public class HelloWorld
{
      public static void main  (String args[] )
      {
            try
            {
            SynthesizerModeDesc test=
new SynthesizerModeDesc(Locale.ENGLISH);
            Voice myvoice  = new Voice();
            test.addVoice(myvoice);

            Synthesizer synth = Central.createSynthesizer(test);
            synth.allocate();
            synth.resume();
            synth.speakPlainText("Hallo Dietmar Hafner", null);
            synth.waitEngineState(Synthesizer.QUEUE_EMPTY);
            synth.deallocate();
            System.exit(0);
            }
            catch (Exception e)
            {
            e.printStackTrace();
            }
      }
```

}Die Programmentwicklung erfolgt mit der Cloud Garden API. Wie bereits erwähnt tut sich diese API schwer bei deutschen Wörtern, da es diese in englischen Deutsch ausspricht.

Die Programmierung der Sprachausgabe wurde danach entscheidend verbessernd die Parameter für ein „angenehmklingende Sprache“ festgelegt.

Parametrisierung für die Sprache:

Grundsätzlich kommt ein eigens angepasstes Sprachmodell zum Einsatz, welches man sich selbst zusammenstellen kann. Wie bereits zuvor erwähnt bietet JSML einige Möglichkeiten, die Sprachqualität zu verbessern.

Weitere Möglichkeiten sich eine „angenehme“ Sprachwiedergabe zu basteln bietet die Eigenschaften sich die Sprach- Engine entsprechend einzustellen.

Bei der Erzeugung einer Sprach Engines, kommen folgende Zeilen zum Einsatz:

```
      SynthesizerModeDesc desc = new SynthesizerModeDesc(Locale.ENGLISH);
synth = Central.createSynthesizer(desc);
desc = (SynthesizerModeDesc)synth.getEngineModeDesc();
```

Dem Synthesizer, welcher oben beschrieben wurde, können nun bestimmte Eigenschaften hinzugefügt werden.

```
SynthesizerProperties props = synth.getSynthesizerProperties();
Voice myvoice = new Voice("Male Mittleres Alter", Voice.GENDER_MALE,
Voice.AGE_MIDDLE_ADULT, null);
props.setVoice(myvoice);
props.setSpeakingRate(150.0f);
```

Es kann sich eine “eigene” Sprache definiert werden, wo Parameter, wie Geschlecht und Alter einfließen. Zu beachten ist, das bei der API von Cloud Garden nicht alle spezifizierten Altersgrößen verwirklicht wurden. Enthalten sind: Age = Child, Teen, Adult oder Senior (wobei alle Sprachen von der SAPI auf Adult gesetzt werden)

Vordefinierte Spracheigenschaften der SAPI können auch verwendet werden.

Es stehen drei zur Auswahl: Microsoft Mike, Microsoft Sam oder Microsoft Mary.

In meinen Programmteil, wurde die Sprache von „Microsoft Mike“ eingesetzt, da sie sehr gut verstanden werden konnte. (Man muss jedoch beachten, dass in dieser JSAPI Version, kein R enthalten ist und deutsche Wörter, englisch ausgesprochen werden)

Sprachumsetzung

Dafür ist die SAPI 5.0 von Microsoft zuständig. Für Windowsplattformen, stellt sie die Verbindung von einem Programm (C++ oder JAVA) zu der Hardware her. Dies ist wiederum praktisch, da man sich um hardwarenahe Programmierung somit keine Gedanken machen muss.

Das Interface von einen JAVA Programm zur Hardware wird auch JNI (JAVA Native Interface) genannt.

Es werden nun die Code Teile der Datei SpeakableAuftrag.java gezeigt.

Sprachprogrammteil:

```
      /* Diese Funktion erzeugt eine Sprach Engine, ENGLISH,
      Microsoft Mikes Stimme
      *Wort pro Minute Rate 170, Volume ganz aufgedreht.
      *somit auch den Speicherbereich*/

      public Synthesizer CreateSynthesizer()
      {

            Synthesizer synth = null;
            try
            {
              SynthesizerModeDesc desc =
                  new SynthesizerModeDesc(Locale.ENGLISH);
             synth = Central.createSynthesizer(desc);
             desc = (SynthesizerModeDesc)synth.getEngineModeDesc();
```

```
                /* Nun wird Speicher bereitgestellt*/
            synth.allocate();

            /*Weitermachen nicht stoppen */
            synth.resume();

            /* Status des Engines auf ALLOCATED setzen */
            synth.waitEngineState(Synthesizer.ALLOCATED);

              /*Eigenschaften bearbeiten*/
              SynthesizerProperties props =
                  synth.getSynthesizerProperties();
                  Voice myvoice  = new Voice("Microsoft Mike",
                        Voice.GENDER_MALE, Voice.AGE_MIDDLE_ADULT, null);
                  props.setVoice(myvoice);
                  props.setVolume(1.0f);
                  props.setSpeakingRate(170.0f);

            }
            catch (Exception e1)
            {
                    System.out.println(e1);
            }
            return synth;
      }
      /* Bekommt einen bereits allokierten Synthesizer sowie den String
      welcher gesprochen werden muss.*/
      public void SpeakString(Synthesizer synth, String StSpeak)
      {

            try
            {
                  /* Sprachausgabe des jeweiligen Strings*/
            synth.speak(StSpeak,null);

               synth.waitEngineState(synth.QUEUE_EMPTY);
            }
            catch (Exception e2)
            {
                  System.out.println(e2);
      }
      }
      public void CloseSynth(Synthesizer synth)
      {
            try
            {  synth.deallocate();
                  synth.waitEngineState(synth.DEALLOCATED);
            }
            catch (Exception e1)
            {
                  System.out.println("API wurde heruntergefahren");
            }
            }
```

4.2 Graphische Programmierung

JAVA wurde von Anfang an so entwickelt, dass eine graphische Oberfläche zur Verfügung gestellt werden kann. Die gesamte Graphikbibliothek wird als Abstract Windowing Toolkit (AWT) zusammengefasst.

Die Fähigkeiten von AWT lässt sich in 4 Gruppen unterteilen:

- Einfache graphische Operationen zum Zeichnen von Linien oder Füllen von Flächen und zur Ausgabe von Text.
- Eventhandling für den Programmablauf (Maus-, Tastatur- und Fensterereignisse)
- Dialogboxen
- Darstellung von Bitmaps und Ausgabe von Sound

Ziel ist es auch graphische Anwendungen auf verschiedene Plattformen annähernd ähnlich ablaufen zu lassen.

Besondere Eventverarbeitung ist notwendig um die Performance von JAVA Programmen nicht zu mindern.

4.2.1 Programmteil:

```
	public static void main(String [] agrs)
	{
		SpeakableAuftrag window= new SpeakableAuftrag(); // Objekt
		erzeugen vom Konstruktor der Klasse
		window.setVisible(true);
	}
	public SpeakableAuftrag()
	{
		super("Sprachverarbeitssystem Hafner Version 1.1");
		// Titelzeile
		setBackground (Color.white);
		// Hintergrundfarbe
		setSize (400,300);
				//Größe des Fensters
		setLayout(null);
				// Kein Layout übernehmen

		Button breplay = new Button("Sprechen");
		breplay.setBounds (40,180,60,30);
		breplay.addActionListener(this);

		Button bclose = new Button("Close");
	//Button erzeugen
		bclose.setBounds ( 270,180,50,30);
		bclose.addActionListener(this);
```

```
            add(bclose);
            add(breplay);


      }
      /*Diese Funktion zeichnet auf den Frame mit den angegeben Koordinaten
      *die entsprechenden String */
      public void paint (Graphics gtest)
      {

      gtest.drawString("Sprachwiedergabe des aktuellen Auftrags",70,50);
      gtest.drawString("Verbindung zu den Aufträgen erfolgt über die
Datenbank",68,65);
      gtest.drawString("Per Sprache werden ANZAHL, NAME und TISCH
ausgegben" ,30, 120);

      }


      /* Beim Drücken auf den Close Button, sowie anderer Aktionen werden
folgende
      * Programmteile ausgeführt*/
      public void actionPerformed (ActionEvent e)
      {
            String Ausgabe;
            String cmd = e.getActionCommand ();
            if(cmd.equals("Close"))
            {
                  Synthesizer mysynth=null;
                  //mysynth1=CreateSynthesizer();
                  CloseSynth(mysynth);
                  //setVisible(false);
                  //dispose();
                  System.exit(0);

            }
            if (cmd.equals("Sprechen"))
            {


            try
            {
                  Ausgabe=getAuftragsTabelle();
                  Synthesizer mysynth=null;
                  mysynth=CreateSynthesizer();
                  SpeakString(mysynth, Ausgabe);
                  System.out.println("Sprache ausgegeben");
            }
            catch (Exception e4)
            {
                        System.out.println("Keine Daten von der Tabelle
oder Fehler beim Initialieren");
            }
```

```
            }

        }
```

Die Applikation sieht folgendermaßen aus:

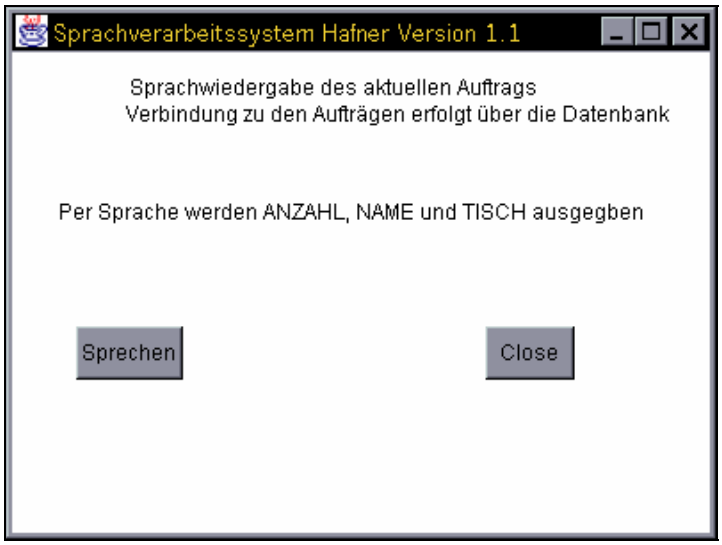

Abbildung 9

Nach Klicken auf „Sprechen“ wird eine Verbindung zur Datenbank hergestellt. Die Datensatz aus der Tabelle AUFTRAG wird ausgelesen. Dieser wird zurückgegeben und dann gesprochen.

4.3 Datenbankzugriff:

Da die Daten der Aufträge in der Tabelle „Auftrag“ enthalten sind, muss auf diese zugegriffen werden, da diese verarbeitet werden müssen. Es wird bei der SQL Query SELECT * from AUFTRAG ein String zusammengebaut, der bereits mit JSML ausgestattet wird und schlussendlich gesprochen wird.

In unserem Jahreslaborprojekt wurde schon bereits die MySQL Datenbank eingesetzt. Einige Programmteile konnte wiederverwendet werden.

Um diese mit JAVA nutzen zu können, müssen die Datenbank Treiber von MySQL verwendet werden.

Folgend ging man vor:

Das JAVA Programm wurde im Verzeichnis ausgeführt, auf welcher der CLASSPATH zeigt. Weiters wurden entsprechende Dateien in dieses Verzeichnis kopiert.

4.3.1 Programmteil:

```
public class SpeakableAuftrag extends Frame
implements ActionListener
```

```
{
   static String driver = "org.gjt.mm.mysql.Driver"; //MySQL-JDBC-Treiber
      static String url = "jdbc:mysql://127.0.0.1/WAPSCAS"; //MySQL-DB-Url
      static String user = "root"; //MySQL-Username
      static String passwd = ""; //MySQL-Passwort
      …
      …
}
      /* Stellt die Verbindung zur Datenbank her und gibt ein ResultSet
            Objekt zurück */
      public static ResultSet getResultSet(String query)
      throws SQLException, ClassNotFoundException
      {
            try
            {
                  Class.forName (driver);
                  Connection con =
                  DriverManager.getConnection(url,user,passwd);
                  Statement stmt = con.createStatement();
                  ResultSet rs = stmt.executeQuery(query);
                  con.close();
                  stmt.close();
                  return(rs);
            }
            catch (ClassNotFoundException cnfe)
            {
                  System.out.println(cnfe);
                  return null;
            }
            catch (SQLException se)
            {
                  System.out.println(se);
                  return null;
            }

      }



      /* Liefert einen String zurück, der bereits gesprochen werden kann
(enthält JSML).
      * und Auftragbestätigungsbutton zurück.
      */
      public static String getAuftragsTabelle()
            throws SQLException, ClassNotFoundException
      {
            try
            {
            String query = "SELECT * FROM Auftrag WHERE fertig = 0;", name;
            //SELECT QUERY
            int auftrag_nr, anzahl, tisch_nr, kellner_nr;
            StringBuffer auftrag = new StringBuffer();
            ResultSet rs = getResultSet(query);
            //Rückgabe des aktuellen Result Sets
            name = rs.getString("name");
            anzahl  = rs.getInt("anzahl");
```

```
            tisch_nr = rs.getInt("tisch_nr");

            while (rs.next())
            {}
            /*Ein großer String wird gebaut, der gesprochen wird */
            auftrag.append("<emp>Anzahl<break size=\"large\" />"+anzahl+
       "<break size=\"large\" />Name <break size=\"large\" />"+name+
       "<break size=\"large\" /> Tisch <break size=\"large\" />"+tisch_nr+
       "</emp>");
       System.out.println(anzahl +" "+ name + " " + tisch_nr);
      System.out.println("Daten von der Datenbank erfolgreich erhalten");
            return (auftrag.toString());

            }
            catch (ClassNotFoundException cnfe)
            {
                  System.out.println(cnfe);
                  return null;
            }
            catch (SQLException se)
            {
                  System.out.println(se);
                  return null;
            }
      }
```

4.4 Ressourcenbedarf

Es werden zunächst Hardwareressourcen vorrausgesetzt, um JAVA Speech überhaupt nutzen zu können (siehe Punkt 2.7)

In weiterer Folge, ist jedoch auch die Auslastung in Sachen Speicher und Prozessor interessant.

Mit dem Systemmonitor wurden ein paar Versuche durchgeführt und es zeigt sich, dass beim Laden des JAVA Programms, die Speicherauslagerung um 6MB anstieg. Wird nun für diese Sprach Engine Speicher reserviert, so werden insgesamt 20 MB ! angebraucht. Dies ist rein zur dynamischen Sprachwiedergabe.

Die Prozessorauslastung wurde ebenfalls betrachtet (PC System mit 1000MHz). Man kann sagen, das kein Unterschied zu sehen war, ob nun gesprochen wurde oder nicht. (3-5% Schwankungen erkennbar). Für die Initialisierung der Anwendung werden 20% Prozessorauslastung verwendet.

5 Schlussfolgerungen

Bei der Programmerstellung kam es zu einigen Problemen, die nun folgend erläutert werden:

Am Anfang ergab sich das Problem mit den JAVA Speech APIs. Es standen mehrere zur Auswahl, doch nur eines davon funktionierte einwandfrei. Um die APIs doch zum laufen zu

bekommen wurden auch verschiedene Versionen der JDK verwendet. Letztlich war die JDK 1.2.2 eine guter „Kompromiss“.

Die Cloudgarden API wurde somit für die JAVA Speech Programmierung eingesetzt. Zu bemerken ist noch, dass die Microsoft SAPI 5.0 installiert werden muss, um diese API zu verwenden. Diese SAPI benötigt jedoch mehr als 150 MB Speicherplatz, wobei das JAR File der API lediglich 100kByte benötigt.

Die zweite API, die probiert wurde, war jene von IBM. Diese stürzte jedoch mit folgender Fehlermeldung immer ab.

Abbildung 10

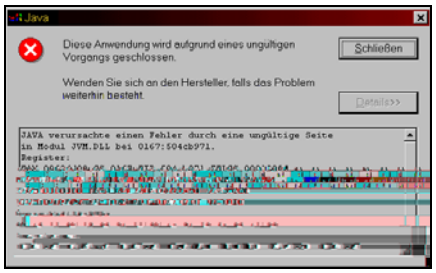

Abbildung 11

Das Problem lag darin, dass bei der Microsoft JAVA Virtual Machine, die keine JNI unterstützte. Eine JNI stellt die Verwendung zur Hardware des Systems her. Doch auch mit Verwendung der Microsoft SAPI, stürzte diese API immer wieder ab.

Bei den anderen API kamen ähnliche Fehlermeldungen, schon beim Kompilieren des Sourcecodes.

Anzumerken ist noch, dass die CloudGarden API auch abstürzt, doch erst am Ende nach erfolgreicher Sprachausgabe.

Probleme gab es auch bei der Verwendung der API als Applet (siehe Lösungswege). Es wird zwar beschrieben, wie man vorzugehen hat, wenn man ein Applet einsetzt, doch auch dies schien nicht zu funktionieren. Es gibt auch etwas positives daran, weil nicht mehr darauf aufgepasst werden muss, das Benutzerressourcen verwendet werden. Signierte Applets (Applets, die auf Benutzerressourcen zugreifen dürfen) schienen auch nicht als sehr sinnvoll.

Darum wurde eine graphische Applikation erstellt, die diese Schwierigkeiten nicht hat.

Als Schlussfolgerung dieser Probleme, kann die Verzögerung mit der Arbeit der Diplomarbeit angesehen werden.

Es war folglich keine Zeit mehr, die Spracherkennung einzubauen, da bereits bei Versuchen mit dem doppelten Zeitaufwand zu rechnen war.

6 Literaturverzeichnis

[1] JAVA Speech Grammar Format Specification
http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/index.html

[2] IBM's Speech for JAVA API
http://www.alphaworks.ibm.com/tech/speech

[3] Jesse Petersons JAVA Speech API
http://www.jpeterson.com/rnd/

[4] The Cloudgarden JAVA Speech API
http://www.cloudgarden.com/

[5] Microsoft Speech.net Technologies
http://www.microsoft.com/speech/

[6] Implementation von Eric Hambuch
http://www.stud.fernuni-hagen.de/q3988600/java/javaspeech.html

[7] Java Speech API Programmer's Guide (in ausgedruckter Form)
http://java.sun.com/products/java-media/speech/forDevelopers/jsapi-guide/index.html

[8] Java Speech Markup Language Specification (in ausgedruckter Form)
http://java.sun.com/products/java-media/speech/forDevelopers/JSML/index.html

[9] GotoJava 2 (lokal in HTML Form)
http://www.javabuch.de/